

Airline Passenger Satisfaction Prediction

Author: Michael Holthouser



Business Understanding

Explorer Airlines has tasked me with conducting a predictive analysis of their passengers' satisfaction levels and identifying key features that have the greatest influence on customer satisfaction. In the past year, Explorer Airlines has observed a decline in customer satisfaction, and the board of executives wants to understand what the company needs to address to halt this downward trend.

During the pandemic, travelers enjoyed some unexpected benefits, such as fewer passengers, emptier flights, and ticket price flexibility. This led to more space on airplanes, reduced waiting times, and increased attention from flight attendants. Now that the pandemic is behind us, travelers are no longer afraid of flying, and flights are once again fully occupied.

While some may view the decrease in satisfaction as a negative development, it can also be seen as an opportunity. Michael Taylor, travel intelligence lead at J.D. Power said it best, "If airlines can find ways to manage the growing passenger volumes and make small adjustments to help

passengers feel more valued, they should be able to overcome this challenge and return to

Evaluation Metric : F1-Score

- F1 score is a harmonic mean of precision and recall, which are both important metrics in classification tasks. Precision measures the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positive predictions among all actual positives.
- In predicting customer satisfaction for airline passengers, both precision and recall are equally important. A high precision means that the model is making fewer false positive predictions, or in other words, correctly identifying satisfied customers. A high recall means that the model is making fewer false negative predictions, or in other words, correctly identifying dissatisfied customers.
- F1 score balances the tradeoff between precision and recall by taking into account both metrics. A high F1 score means that the model is performing well in both identifying satisfied and dissatisfied customers. Therefore, using F1 score as the evaluation metric ensures that the model is performing well in both identifying satisfied and dissatisfied customers, which is important for predicting customer satisfaction for airline passengers.

Models

- For this project I used 5 different classification models:
 1. **Logistic Regression** - I used logistic regression as my baseline model. After splitting the data into train and test sets, I checked for imbalance. The result was that the data was pretty well balanced. So imbalancing techniques like SMOTE were not necessary. Logistic Regression was my worst performing model, and did not score as high as the other models.
 - Roc auc score = 89%
 - Neutral or Dissatisfied F1-score = 91%
 - Satisfied F1-score = 87%
 - Misclassified Predictions = 2534
 2. **Decision Tree** - I performed two different decision tree models: decision tree with no tuning, and a decision tree using GridSearchCV. Decision trees require a lot of tuning to become accurate, that is why after my first model with no tuning showed signs of overfitting. After performing a gridsearch on my second model, I ran the model again but with more hyperparameters to try to improve accuracy and reduce overfitting. Compared to the logistic regression model, the decision tree with gridsearch performed much better at predicting false negatives, false positives, and false negatives.
 - Roc auc score = 95%
 - Neutral or Dissatisfied F1-score = 96%
 - Satisfied F1-score = 94
 - Misclassified Predictions = 1140
 3. **Random Forest** - Like the decision tree, I ran two different random forest models. One with no tuning, and other other using GridSearchCV. The model without tuning, like the decision tree,

showed signs of overfitting. After adding gridsearch to the model, the evaluation metrics were excellent. However, since random forest doesn't perform quite as well with balanced data, the results of the confusion matrix wasn't ideal. It did not do well at predicting false negatives, false positives, and false negatives.

- Roc auc score = 94%
- Neutral or Dissatisfied F1-score = 95%
- Satisfied F1-score = 93%
- Misclassified Predictions = 1398

4. **CatBoost** (best model) - Catboost performed very well on the data. Catboost was designed to work well with categorical data and has built-in functionality for performing cross validation, which helps prevent overfitting. I ran a model with no tuning, and then a second catboost model using RandomSearchCV. Random search is similar to gridsearch, in the sense it picks the hyperparameters to help the performance of the model. However, it does not do a exhaustive search making it less computationally expensive. But since the model generally performs well without tuning, the two models both performed very well.

- Roc auc score = 96%
- Neutral or Dissatisfied F1-score = 97%
- Satisfied F1-score = 95%
- Misclassified Predictions = 924

5. **XGBoost** (Best Model)- XGBoost is another high performance machine learning model that performed very well with the data. Although, the scores were very similar to the catboost model and could have easily been chosen as the best model. Like catboost, I ran two xgboost models. One with no tuning and one using random search. It performed well at reducing the number of false positives and false negatives, as well as predicting true negatives.

- Roc auc = 96%
- Neutral or Dissatisfied F1-score = 97%
- Satisfied F1-score = 95
- Misclassified Predictions = 918

Relevant Packages and Libraries

In [1]:

```

1 # data manipulation and analysis
2 import pandas as pd
3 import numpy as np
4
5 # visualization
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 sns.set()
9
10 # preprocessing and transformation
11 from sklearn.preprocessing import LabelEncoder, MinMaxScaler, OrdinalEncoder
12 from sklearn.compose import ColumnTransformer
13
14 # model selection and evaluation
15 from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
16 from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix
17 from sklearn.metrics import roc_curve, auc
18
19 # machine learning algorithms
20 from sklearn.ensemble import RandomForestClassifier
21 import xgboost as xgb
22 from sklearn.linear_model import LogisticRegression
23 from sklearn.tree import DecisionTreeClassifier
24 import catboost as ctb
25
26 # Imbalanced data handling
27 from imblearn.pipeline import Pipeline
28 from sklearn.pipeline import Pipeline
29
30 # Miscellaneous
31 from scipy.stats import randint as sp_randint
32 import warnings
33 warnings.filterwarnings("ignore")
34
35 #set display options
36 pd.set_option('display.max_columns', None)

```

Data Understanding

- The data was downloaded from [Kaggle](https://www.kaggle.com/datasets/mysarahmadbhat/airline-passenger-satisfaction) (<https://www.kaggle.com/datasets/mysarahmadbhat/airline-passenger-satisfaction>). The dataset is comprised of 103,904 airline passenger surveys about their travel experience. The dataset was comprised mostly of categorical data, and the target variable was 'satisfaction'. Satisfaction was divided up into two options making this a binary classification problem. Passengers were classified as either being 'satisfied' or 'neutral/dissatisfied'.
- The dataset is mostly comprised of categorical data.
 - The survey columns that were rated from 1-5 are known as ordinal categorical data.
 - The columns that have only two options are known as binary categorical data.

- The columns that have categorical data that don't have any order or hierarchy, such as gender, customer type, and class are known as nominal.

In [2]:

```
1 #import the data
2 data = pd.read_csv("data/train.csv")
3 data.head()
```

Out[2]:

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenience
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	

Column Descriptions:

- **Gender:** Gender of the passengers (Female, Male)
- **Customer Type:** The customer type (Loyal customer, disloyal customer)
- **Age:** The actual age of the passengers
- **Type of Travel:** Purpose of the flight of the passengers (Personal Travel, Business Travel)
- **Class:** Travel class in the plane of the passengers (Business, Eco, Eco Plus)
- **Flight distance:** The flight distance of this journey
- **Inflight wifi service:** Satisfaction level of the inflight wifi service (0:Not Applicable;1-5)
- **Departure/Arrival time convenient:** Satisfaction level of Departure/Arrival time convenient
- **Ease of Online booking:** Satisfaction level of online booking
- **Gate location:** Satisfaction level of Gate location
- **Food and drink:** Satisfaction level of Food and drink
- **Online boarding:** Satisfaction level of online boarding
- **Seat comfort:** Satisfaction level of Seat comfort
- **Inflight entertainment:** Satisfaction level of inflight entertainment
- **On-board service:** Satisfaction level of On-board service
- **Leg room service:** Satisfaction level of Leg room service
- **Baggage handling:** Satisfaction level of baggage handling
- **Check-in service:** Satisfaction level of Check-in service
- **Inflight service:** Satisfaction level of inflight service
- **Cleanliness:** Satisfaction level of Cleanliness
- **Departure Delay in Minutes:** Minutes delayed when departure
- **Arrival Delay in Minutes:** Minutes delayed when Arrival
- **Satisfaction :** Airline satisfaction level (Satisfaction, neutral or dissatisfaction)

Data Cleaning

- Drop unnecessary columns: Unnamed, id
- Format columns to use conventional python coding. Words should be lower-case and use snake case.
- Remove rows with NaN values.
- Rename elements in 'Customer Type' for easier comprehension.
- Rename certain columns for easier comprehension.
- The survey rows that contain satisfaction scores of 0 will be removed.
 - The customer most likely did not indicate the score for that particular category.
- Columns, Departure Delay in Minutes and Arrival Delay in Minutes will be combined into a column called Total Delay.
- Our target variable will be changed from an object data type to a numerical data type.

Drop Columns: Unnamed, id

- Unnamed and id are not necessary for our analysis. Therefore, I will remove them from the dataset.

```
In [3]: 1 data = data.drop(['Unnamed: 0', 'id'], axis=1)
2 data.head(1)
```

Out[3]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location
0	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	3	1

Rename Columns

- Next I will rename the columns to follow the conventional python developer format.
 - lowercase and snake case.

```
In [4]: 1 # Function to convert strings to snake case
2 def snake_case(string):
3     return string.lower().replace(' ', '_')
4
5 #rename columns to lower case using snake case
6 for column in data.columns:
7     data = data.rename(columns={column: snake_case(column)})
```

In [5]:

```

1 #Call info() method to check columns names, dtypes, and number of records
2 data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   gender          103904 non-null   object  
 1   customer_type   103904 non-null   object  
 2   age              103904 non-null   int64  
 3   type_of_travel  103904 non-null   object  
 4   class            103904 non-null   object  
 5   flight_distance 103904 non-null   int64  
 6   inflight_wifi_service 103904 non-null   int64  
 7   departure/arrival_time_convenient 103904 non-null   int64  
 8   ease_of_online_booking 103904 non-null   int64  
 9   gate_location    103904 non-null   int64  
 10  food_and_drink  103904 non-null   int64  
 11  online_boarding 103904 non-null   int64  
 12  seat_comfort    103904 non-null   int64  
 13  inflight_entertainment 103904 non-null   int64  
 14  on-board_service 103904 non-null   int64  
 15  leg_room_service 103904 non-null   int64  
 16  baggage_handling 103904 non-null   int64  
 17  checkin_service  103904 non-null   int64  
 18  inflight_service 103904 non-null   int64  
 19  cleanliness      103904 non-null   int64  
 20  departure_delay_in_minutes 103904 non-null   int64  
 21  arrival_delay_in_minutes 103594 non-null   float64 
 22  satisfaction     103904 non-null   object  
dtypes: float64(1), int64(17), object(5)
memory usage: 18.2+ MB

```

Observations:

- The info function shows the columns, the data types of each column, and the number of records that each column has.
- The data set containing 103904 rows and 23 columns.
- There is only one column with missing data, arrival_delay_in_minutes.
- Satisfaction is our dependant variable for our classification models.

Descriptive Statistics

- Printing out descriptive statistics is a good practice to have a basic understanding of the data.
- It provides a summary of the data distribution, including measures of central tendency (such as mean and median) and measures of variability (such as standard deviation and range), as well as information about missing values. - - Descriptive statistics can help to identify potential issues with the data, such as outliers or data entry errors.
- Descriptive statistics can be useful for comparing different datasets or subsets of the same dataset.

In [6]: 1 data.describe()

Out[6]:

	age	flight_distance	inflight_wifi_service	departure/arrival_time_convenient	ease_of_online_booking
count	103904.000000	103904.000000	103904.000000		103904.000000
mean	39.379706	1189.448375	2.729683		3.060296
std	15.114964	997.147281	1.327829		1.525075
min	7.000000	31.000000	0.000000		0.000000
25%	27.000000	414.000000	2.000000		2.000000
50%	40.000000	843.000000	3.000000		3.000000
75%	51.000000	1743.000000	4.000000		4.000000
max	85.000000	4983.000000	5.000000		5.000000

Check For Missing Values

In [7]: 1 # Check for miss values and sum them up
2 data.isna().sum()

Out[7]:

gender	0
customer_type	0
age	0
type_of_travel	0
class	0
flight_distance	0
inflight_wifi_service	0
departure/arrival_time_convenient	0
ease_of_online_booking	0
gate_location	0
food_and_drink	0
online_boarding	0
seat_comfort	0
inflight_entertainment	0
on-board_service	0
leg_room_service	0
baggage_handling	0
checkin_service	0
inflight_service	0
cleanliness	0
departure_delay_in_minutes	0
arrival_delay_in_minutes	310
satisfaction	0
dtype:	int64

- It appears that we have **310** NaN values from the arrival_delay_in_minutes column.
- 310 is not enough missing values to effect my analysis of the data, so dropping them from the dataset is the best option.
- The next step I will drop the 310 records with missing data.

Drop Records With Missing Data

- This code will remove any rows with missing values (NaN) in the DataFrame 'data'. The parameter axis=0 specifies that it will drop rows (as opposed to columns),

```
In [8]: 1 # drop rows with missing data
2 data.dropna(axis=0, inplace=True)
```

```
In [9]: 1 # Check to see if missing data records have been removed.
2 data.isna().sum()
```

```
Out[9]: gender          0
customer_type      0
age                0
type_of_travel     0
class              0
flight_distance    0
inflight_wifi_service 0
departure/arrival_time_convenient 0
ease_of_online_booking 0
gate_location       0
food_and_drink      0
online_boarding     0
seat_comfort        0
inflight_entertainment 0
on-board_service    0
leg_room_service    0
baggage_handling    0
checkin_service     0
inflight_service    0
cleanliness         0
departure_delay_in_minutes 0
arrival_delay_in_minutes 0
satisfaction        0
dtype: int64
```

- All rows with missing data have been removed.

Rename 'customer_type' Elements

- I opted to rename the values within the customer_type column for better understanding.
- I thought returning customer and first-time customer made more sense when thinking about the data.

```
In [10]: 1 # change the names of the customer type elements
2 data['customer_type'] = data['customer_type'].map({'Loyal Customer': 'Re
3 'disloyal Customer':
```

Rename column names

- Similar to customer_type, I renamed the elements within the column to make better sense, when I was thinking about the data.
- Changing the name should also make more sense to those who are interested in the analysis of the notebook.

```
In [11]: 1 # change the name of columns to make more sense
2 data = data.rename(columns={'leg_room_service':'leg_room',
3                           'departure/arrival_time_convenient':'depart
```

```
In [12]: 1 # Check to see if changes have been made to the dataframe
2 data
```

Out[12]:

		gender	customer_type	age	type_of_travel	class	flight_distance	inflight_wifi_service	distance
0	Male	Returning Customer	13	Personal Travel	Eco Plus	460		3	
1	Male	First-time Customer	25	Business travel	Business	235		3	
2	Female	Returning Customer	26	Business travel	Business	1142		2	
3	Female	Returning Customer	25	Business travel	Business	562		2	
4	Male	Returning Customer	61	Business travel	Business	214		3	
...
103899	Female	First-time Customer	23	Business travel	Eco	192		2	
103900	Male	Returning Customer	49	Business travel	Business	2347		4	
103901	Male	First-time Customer	30	Business travel	Business	1995		1	
103902	Female	First-time Customer	22	Business travel	Eco	1000		1	
103903	Male	Returning Customer	27	Business travel	Business	1723		1	

103594 rows × 23 columns

- As you can see, the elements within customer_type have been changed.
- Also columns, leg_room_service has been changed to leg_room, and departure/arrival_time_convenient has been changed to departure/arrival_time_convenience.

Remove rows where the customers did not indicate a satisfaction score

- Scores for these columns should be between 1-5, not 0.
- Passengers who filled out this survey and left a 0 score, did not complete the survey therefore cannot be used for data analysis.

```
In [13]: 1 # Remove rows that contain any zeros
          2 data = data[(data['inflight_wifi_service']!=0) & (data['departure/arrival
          3           (data['ease_of_online_booking']!=0) & (data['gate_location']!=0) &
          4           (data['online_boarding']!=0) & (data['seat_comfort']!=0) &
          5           (data['on-board_service']!=0) & (data['leg_room']!=0) & (data['inflight_meals']!=0) &
          6           (data['inflight_service']!=0) & (data['cleanliness']!=0)]
```

Map target variable, satisfaction column, to contain 0 and 1

- Currently the satisfaction column is made up of two elements of the object data type. I am going to map them to 0 and 1 for modeling purposes.
- **neutral or dissatisfied = 0**
- **satisfied = 1**

```
In [14]: 1 # Change the elements in the satisfaction column to 1 and 0
          2 data['satisfaction'] = data['satisfaction'].map({'neutral or dissatisfied': 0, 'satisfied': 1})
```

Combine columns (departure_delay_in_minutes and arrival_delay_in_minutes) to make total_delay_in_minutes

- The column departure_delay_in_minutes is set as an int data type whereas the arrival_delay_in_minutes is in float data type.
- Before combining the two columns together, I will first change the data type of departure_delay_in_minutes to a float data type.

```
In [15]: 1 # Change data type of departure delay
          2 data['departure_delay_in_minutes'] = data['departure_delay_in_minutes'].astype(float)
```

```
In [16]: 1 # combine the delay columns to create total_delay_in_minutes
          2 data['total_delay_in_minutes'] = data['departure_delay_in_minutes'] + data['arrival_delay_in_minutes']
```

Drop columns 'departure_delay_in_minutes' and 'arrival_delay_in_minutes'

- Since I created a new column combining the two columns, leaving them in my dataframe is not necessary.

```
In [17]: 1 # drop unnecessary columns
          2 data = data.drop(['departure_delay_in_minutes', 'arrival_delay_in_minutes'])
```

```
In [18]: 1 # Check dataframe for changes
          2 data
```

Out[18]:

		gender	customer_type	age	type_of_travel	class	flight_distance	inflight_wifi_service	de
0	Male	Returning Customer	13	Personal Travel	Eco Plus	460			3
1	Male	First-time Customer	25	Business travel	Business	235			3
2	Female	Returning Customer	26	Business travel	Business	1142			2
3	Female	Returning Customer	25	Business travel	Business	562			2
4	Male	Returning Customer	61	Business travel	Business	214			3
...
103899	Female	First-time Customer	23	Business travel	Eco	192			2
103900	Male	Returning Customer	49	Business travel	Business	2347			4
103901	Male	First-time Customer	30	Business travel	Business	1995			1
103902	Female	First-time Customer	22	Business travel	Eco	1000			1
103903	Male	Returning Customer	27	Business travel	Business	1723			1

95415 rows × 22 columns

- Departure_delay_in_minutes and arrival_delay_in_minutes have been combined to make one column called total_delay_in_minutes.

EDA: Exploratory Data Analysis

- I will first explore the categorical columns, to check their values.
- Then I will label encode them to convert them to a numerical data type for modeling.
- For the visualizations, I have included some of the features that I thought were important with the addition to some of the important features from my final model.

Columns I have chosen to explore:

- **Satisfaction**
- **Gender**
- **Type of travel**
- **Class**
- **Customer type**
- **Seat comfort**

- **Inflight service**
- **Baggage handling**

Show The Unique Values of Each Column

- This code prints out all unique values in each column of the DataFrame data.
- The output shows the distinct values in each categorical variable and helps to identify potential issues such as typos, inconsistent formatting, or categories that need to be grouped together.
- It can also give a sense of the level of detail in the data and help determine appropriate data preprocessing steps.

In [19]:

```
1 for col in data.columns:  
2     print(f'Column: {col}')  
3  
4     print(data[col].unique())
```

```
Column: gender  
['Male' 'Female']  
Column: customer_type  
['Returning Customer' 'First-time Customer']  
Column: age  
[13 25 26 61 47 52 41 20 24 12 53 33 45 38 9 17 43 58 57 49 36 22 31 15  
35 67 37 40 34 39 50 29 54 21 28 27 69 60 23 48 59 46 30 66 64 44 51 32  
19 42 16 11 62 8 56 68 18 55 65 72 70 63 10 7 14 80 74 71 85 73 76 77  
75 79 78]  
Column: type_of_travel  
['Personal Travel' 'Business travel']  
Column: class  
['Eco Plus' 'Business' 'Eco']  
Column: flight_distance  
[ 460 235 1142 ... 974 1479 400]  
Column: inflight_wifi_service  
[3 2 4 1 5]  
Column: departure/arrival_time_convenience  
[4 2 5 3 1]
```

Satisfaction

- The image below shows the distribution of surveys completed with being labeled as satisfied or dissatisfied.
- It appears more people were willing to complete the survey because they were dissatisfied with their travel experience.
- [count/percentages ref \(<https://stackoverflow.com/questions/31749448/how-to-add-percentages-on-top-of-grouped-bars>\)](https://stackoverflow.com/questions/31749448/how-to-add-percentages-on-top-of-grouped-bars)

In [20]:

```
1 # Get value counts and percentages for satisfaction and dissatisfaction
2 print("Satisfaction Counts")
3 print(data['satisfaction'].value_counts())
4 print()
5 print("Percentages")
6 print(data["satisfaction"].value_counts(normalize=True))
```

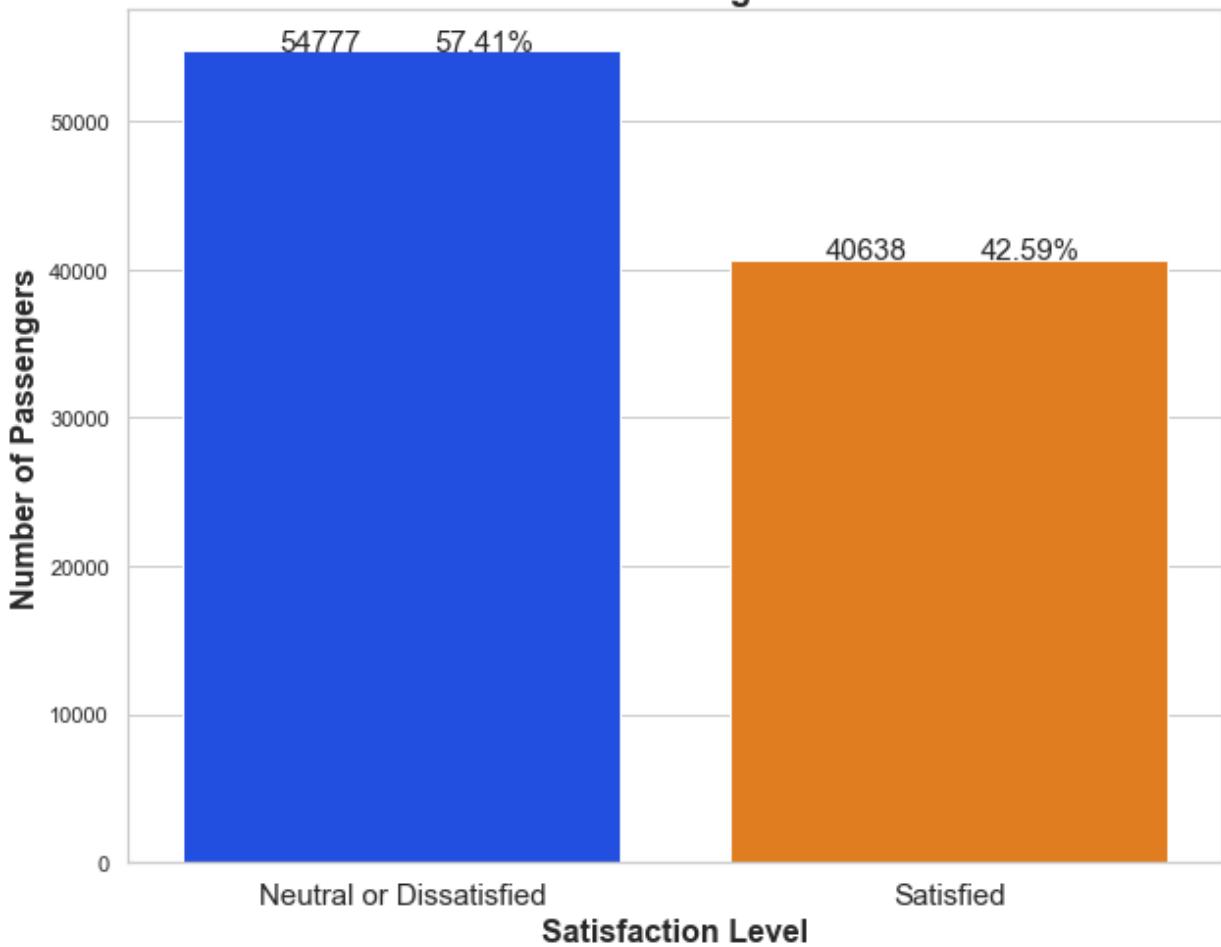
```
Satisfaction Counts
0    54777
1    40638
Name: satisfaction, dtype: int64

Percentages
0    0.574092
1    0.425908
Name: satisfaction, dtype: float64
```

In [21]:

```
1 # set seaborn style to whitegrid
2 sns.set_style('whitegrid')
3 plt.figure(figsize=(10, 8)) # set the figure size
4 # create a count plot of satisfaction and set the color palette to bright
5 ax = sns.countplot(x='satisfaction', data=data, palette='bright')
6 # set the title
7 plt.title('Distribution of Passenger Satisfaction', fontweight='bold',
8 #set the xlabel
9 plt.xlabel('Satisfaction Level', fontweight='bold', fontsize='16')
10 #set the ylabel
11 plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')
12 #set the xticks label
13 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize='14')
14
15 # Add count and percentage labels next to each bar
16 total = float(len(data['satisfaction']))
17 # loop through the value counts of the satisfaction column
18 for i, count in enumerate(data['satisfaction'].value_counts()):
19     percentage = '{:.2f}%'.format((count/total) * 100) # calculate the
20     ax.text(i-0.15, count+50, str(count), ha='center', fontsize='15')
21     ax.text(i+0.15, count+50, percentage, ha='center', fontsize='15') #
22
23 plt.savefig('images/satisfaction.png', format='png') # save the image in
24
25 plt.show()
26
27
28
29
```

Distribution of Passenger Satisfaction



- **Check for imbalance:** With the data being split 57:43 between dissatisfied and satisfied, there is no need for any special resampling techniques to balance the data.
- It is worth noting that the data does weigh heavier in favor of neutral or dissatisfied passengers.

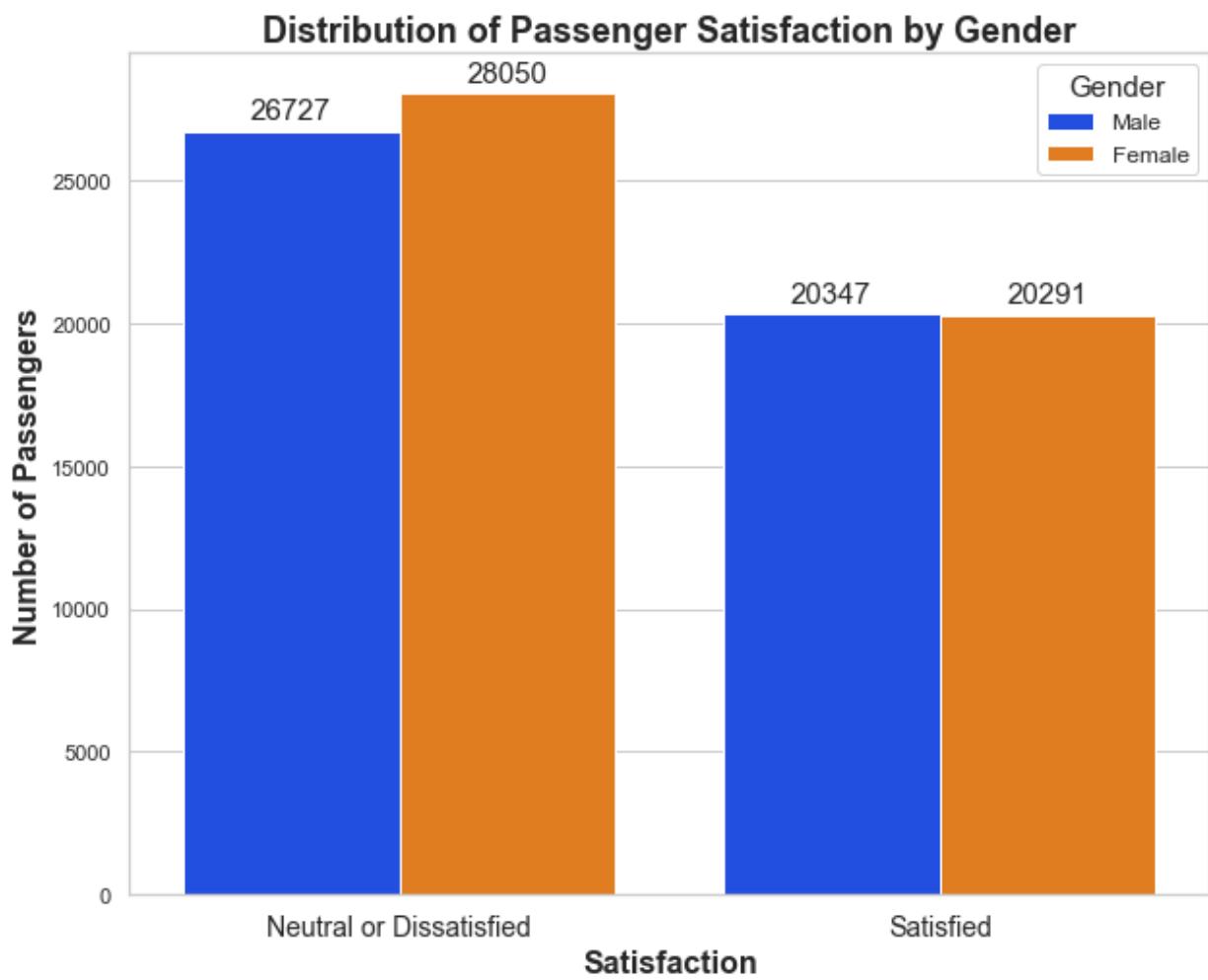
Gender

```
In [22]: 1 # group gender together and get the counts divided up by satisfaction level
          2 gender_group = data.groupby(['satisfaction', 'gender'])['gender'].count
          3 gender_group
```

```
Out[22]: satisfaction  gender
          0             Female    28050
                      Male     26727
          1             Female    20291
                      Male     20347
Name: gender, dtype: int64
```

In [23]:

```
1 # set to white grid style
2 sns.set_style('whitegrid')
3
4 plt.figure(figsize=(10, 8)) # set figure size
5 # create a count plot of satisfaction by gender using the bright color
6 ax = sns.countplot(x='satisfaction', hue='gender', data=data, palette=''
7 #set title
8 plt.title('Distribution of Passenger Satisfaction by Gender', fontweight=
9 plt.xlabel('Satisfaction', fontweight='bold', fontsize='16') # set xlabel
10 plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')# s
11 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=1
12
13 # Add count values to the top of each bar
14 for p in ax.patches:
15     ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width()
16                                     ha = 'center', va = 'center', xytext = (0, 10), textco
17
18 # Increase the size of the legend and its labels
19 plt.legend(fontsize='14', title_fontsize='16')
20
21 # Change the labels of the legend
22 handles, labels = ax.get_legend_handles_labels()
23 ax.legend(handles, ['Male', 'Female'], fontsize='12', title='Gender', t
24
25 plt.savefig('images/gender.png', format='png')
26
27 plt.show();
```



Observations:

- There were more female passengers that were dissatisfied than males.
- The passengers that were satisfied were very close between genders, but with a slight edge in favor of males.
- There could be several reasons why there are more female neutral or dissatisfied passengers.
 - Female passengers are more likely to speak out about their poor experiences.
 - Women are not treated equally while traveling.
 - There simply more women in the dataset than men.

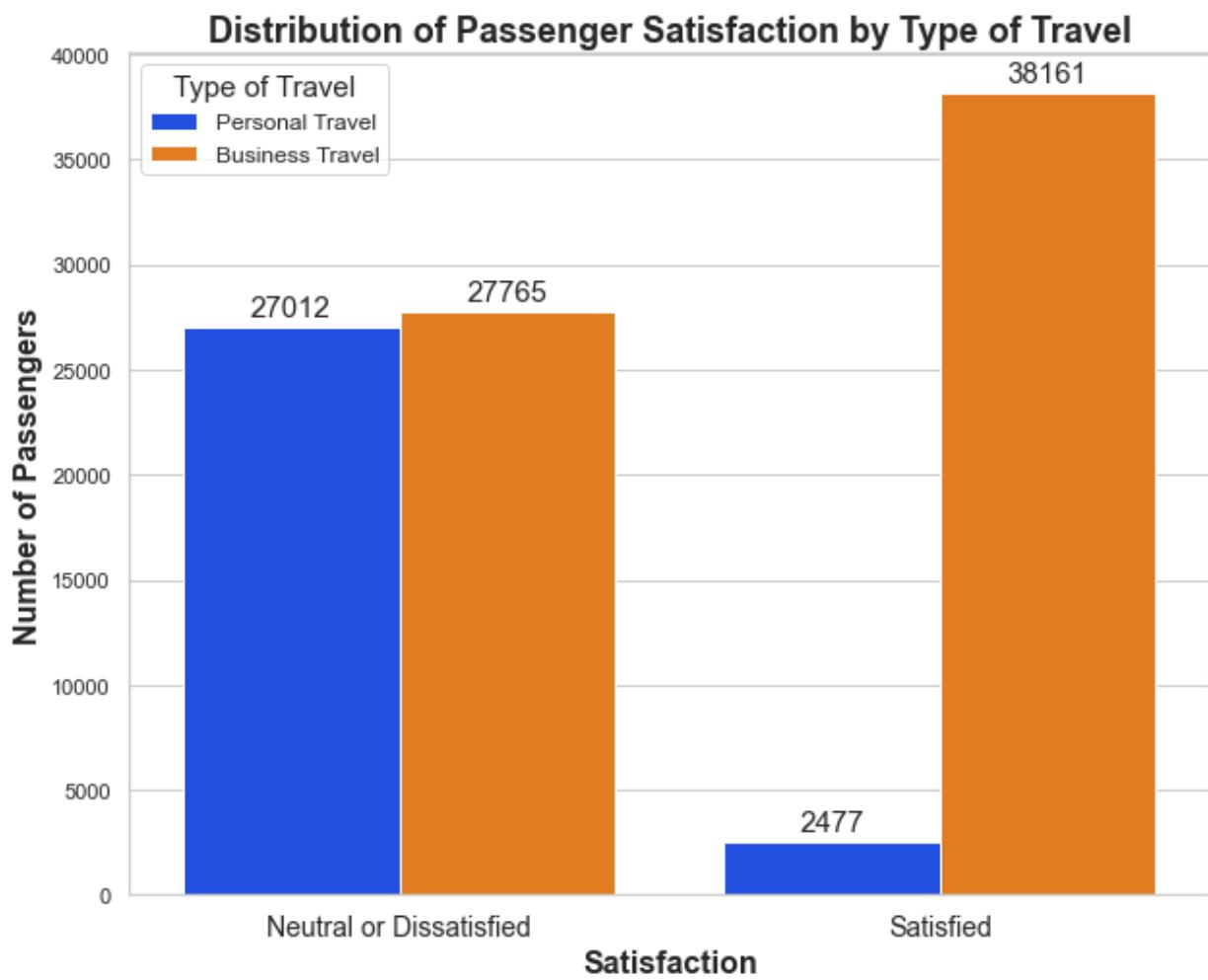
Type of Travel

- **Type of Travel:** What is the purpose for traveling (Personal Travel, Business Travel)
- For business passengers have different needs and expectations than passengers traveling for pleasure. They pay more money for their tickets in return for a more comfortable flight experience.
- Personal travel usually means that the price of travel comes out of the pocket of the person traveling. However, due to the more affordable price of traveling for these particular passengers, the flight experience will not be nearly as comfortable as those traveling for business.

```
In [24]: 1 # count type of travel by satisfaction
          2 type_group = data.groupby(['satisfaction', 'type_of_travel'])['type_of_'
          3 type_group
```

```
Out[24]: satisfaction  type_of_travel
          0             Business travel    27765
                           Personal Travel   27012
          1             Business travel    38161
                           Personal Travel   2477
Name: type_of_travel, dtype: int64
```

```
In [25]: #1set seaborn style to whitegrid
sns.set_style('whitegrid')
#3set figure size
plt.figure(figsize=(10, 8))
#5create a count plot to visualize satisfaction by type of travel. make the chart
ax = sns.countplot(x='satisfaction', hue='type_of_travel', data=data, palette='Set1')
#7set title
plt.title('Distribution of Passenger Satisfaction by Type of Travel', fontweight='bold', fontsize=16)
plt.xlabel('Satisfaction', fontweight='bold', fontsize=16) #set xlabel
plt.ylabel('Number of Passengers', fontweight='bold', fontsize=16) #set ylabel
ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=14)
12
#3Add count values to the top of each bar
for p in ax.patches:
    15    ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2, p.get_y() + p.get_height() / 2),
    16        ha = 'center', va = 'center', xytext = (0, 10), textcoords = 'offset points')
    17
#8Increase the size of the legend and its labels
plt.legend(fontsize='14', title_fontsize='16')
20
#1Change the labels of the legend
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, ['Personal Travel', 'Business Travel'], fontsize='12', title='Type of Travel')
24
plt.savefig('images/type_of_travel.png', format='png')
26
plt.show();
28
```



Observations:

- Looking at this graph in the perspective of airline executives, I would be happy to see the number of satisfied business travelers.
- However, I would be taken back by the number of business travelers who were rated as neutral/dissatisfied. That number should be addressed by the company immediately.
- Business travelers are where airlines make their money. They need to work on reducing the number of neutral or dissatisfied passengers.
 - The average high-yield booking produces 4.3 times more revenue than a typical leisure booking. [OliverWyman \(<https://www.oliverwyman.com/our-expertise/insights/2022/may/airline-economic-analysis-2021-2022.html>\)](https://www.oliverwyman.com/our-expertise/insights/2022/may/airline-economic-analysis-2021-2022.html)

Class

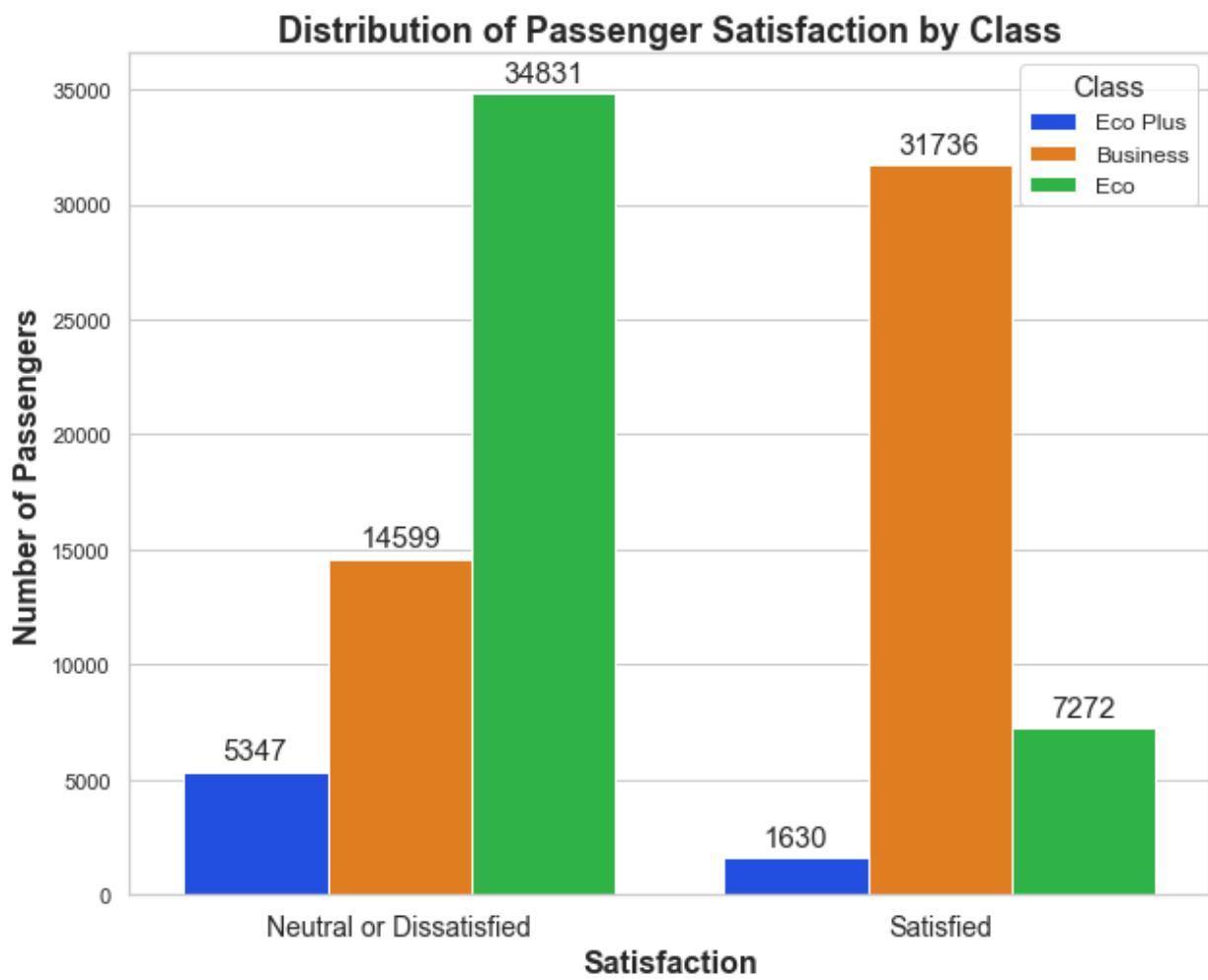
- It is clear that amongst the completed surveys of satisfied passengers, the majority come from business class.
- The clear majority of neutral or dissatisfied passengers were flying in the eco class.

```
In [26]: 1 # group satisfaction by class
          2 class_group = data.groupby(['satisfaction', 'class'])['class'].count()
          3 class_group
```

```
Out[26]: satisfaction  class
          0             Business    14599
                      Eco        34831
                      Eco Plus     5347
          1             Business    31736
                      Eco         7272
                      Eco Plus     1630
Name: class, dtype: int64
```

In [27]:

```
1 # Set seaborn style to whitegrid
2 sns.set_style('whitegrid')
3
4 # set figure size
5 plt.figure(figsize=(10, 8))
6
7 # create a count plot to visualize satisfaction by class. make the colo
8 ax = sns.countplot(x='satisfaction', hue='class', data=data, palette='b
9
10 #Set the title and make it bold
11 plt.title('Distribution of Passenger Satisfaction by Class', fontweight=
12 # set the xlabel and make it bold
13 plt.xlabel('Satisfaction', fontweight='bold', fontsize='16')
14 #set the ylabel
15 plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')
16 #set the xticks label
17 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=1
18
19 # Add count values to the top of each bar
20 for p in ax.patches:
21     ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width(
22                     ha = 'center', va = 'center', xytext = (0, 10), textcoo
23
24 # Increase the size of the legend and its labels
25 plt.legend(fontsize='14', title_fontsize='16')
26
27 # Change the labels of the legend
28 handles, labels = ax.get_legend_handles_labels()
29 ax.legend(handles, ['Eco Plus', 'Business', 'Eco'], fontsize='12', titl
30
31 plt.savefig('images/class.png', format='png') # save image
32
33 plt.show();
```



Observations:

- It is clear that passengers flying economy are the majority of the neutral or dissatisfied group. Business class passengers make up the second most, and the smallest amount of neutral or dissatisfied passengers, come from the economy plus group.
- Business class make up the majority of satisfied passengers, with economy second, and eco plus third.
- Business class passengers make up the majority of the dataset.

Customer Type

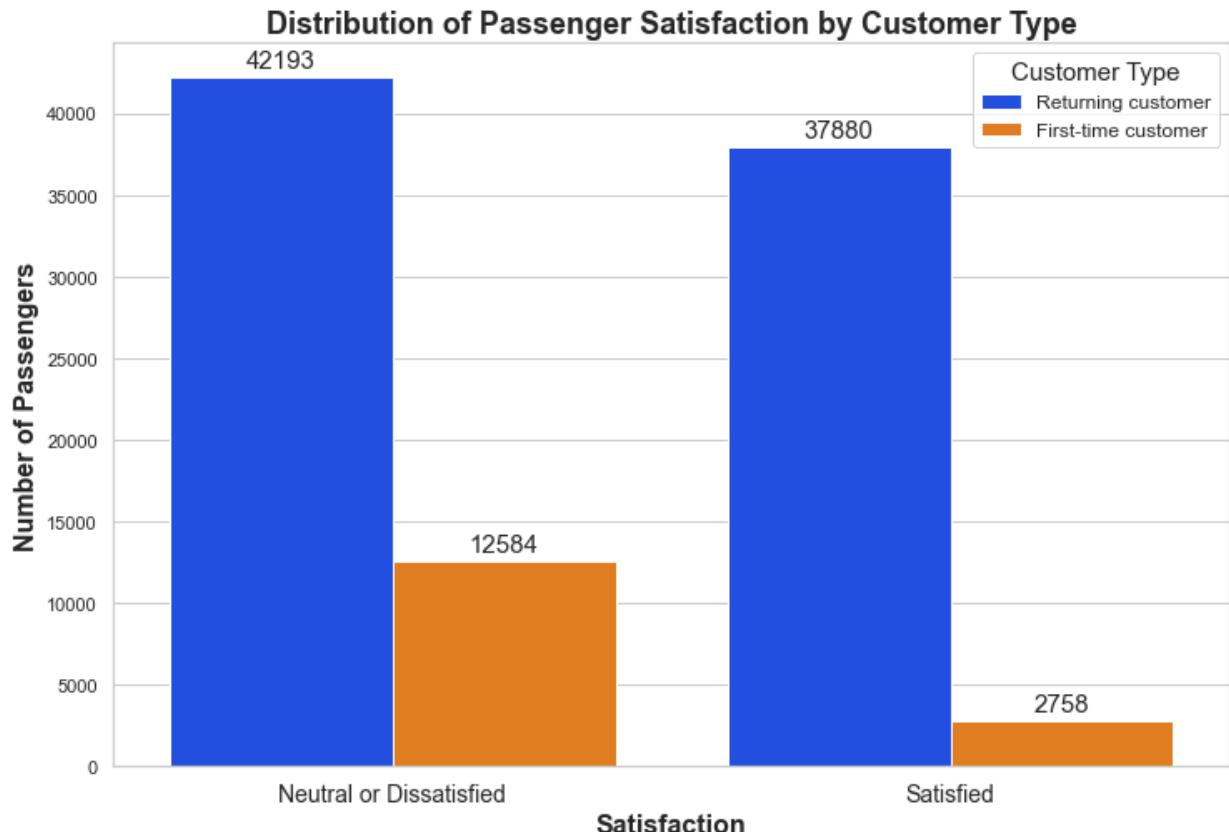
- Returning customers are important for airlines because they tend to be more loyal and profitable.
- They have already experienced the airline's services and are satisfied enough to use them again, which means that the airline has met their expectations.
- Returning customers also tend to be more likely to purchase additional services, such as seat upgrades, additional baggage, or airport lounge access.
- Additionally, they are more likely to recommend the airline to others, which can help to attract new customers. - Retaining existing customers can be more cost-effective than attracting new ones.

```
In [28]: 1 # Group satisfaction by customer type
          2 cust_type_group = data.groupby(['satisfaction', 'customer_type'])['cust
          3 cust_type_group
```

```
Out[28]: satisfaction  customer_type
          0           First-time Customer    12584
                         Returning Customer   42193
          1           First-time Customer    2758
                         Returning Customer   37880
Name: customer_type, dtype: int64
```

- The majority of the dataset is comprised of surveys collected from returning customers.

```
In [29]: # set seaborn style to whitegrid
sns.set_style('whitegrid')
#set the figure size
plt.figure(figsize=(12, 8))
5
#create a count plot to visualize satisfaction by customer type. make the c
ax7 = sns.countplot(x='satisfaction', hue='customer_type', data=data, palett
8
# Set the title and make it bold
plt.title('Distribution of Passenger Satisfaction by Customer Type', fontwe
#Set the xlabel and make it bold
plt.xlabel('Satisfaction', fontweight='bold', fontsize='16')
#Set the ylabel and make it bold
plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')
#Set the xticks labels
ax6.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=14)
17
#Add count values to the top of each bar
for p in ax.patches:
20 ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2,
21 ha = 'center', va = 'center', xytext = (0, 10), textcoords='offset
22
23
#Change the labels of the legend
handles, labels = ax.get_legend_handles_labels()
ax6.legend(handles, ['Returning customer', 'First-time customer'], fontsize=
27
plt.savefig('images/customer_type.png', format='png') #save image
29
plt.show();
```



Observations:

- There were significantly more returning customers in this dataset compared to first-time customers.
- It is good to see that the majority of returning customers were satisfied, but also alarming that there were more returning customers that were classified as unsatisfied.
- Action must be taken to satisfy these loyal customers that had neutral or dissatisfied experiences.

Leg Room

- Leg room and seat comfort go hand in hand. Leg room on a flight directly affects the comfort and satisfaction of customers during their flights.
- When passengers have sufficient leg room, they are more likely to feel comfortable and relaxed, which can lead to a more enjoyable travel experience.
- Conversely, when passengers do not have enough leg room, they may feel cramped, uncomfortable, and restless, which can lead to a negative experience and even physical discomfort or pain.

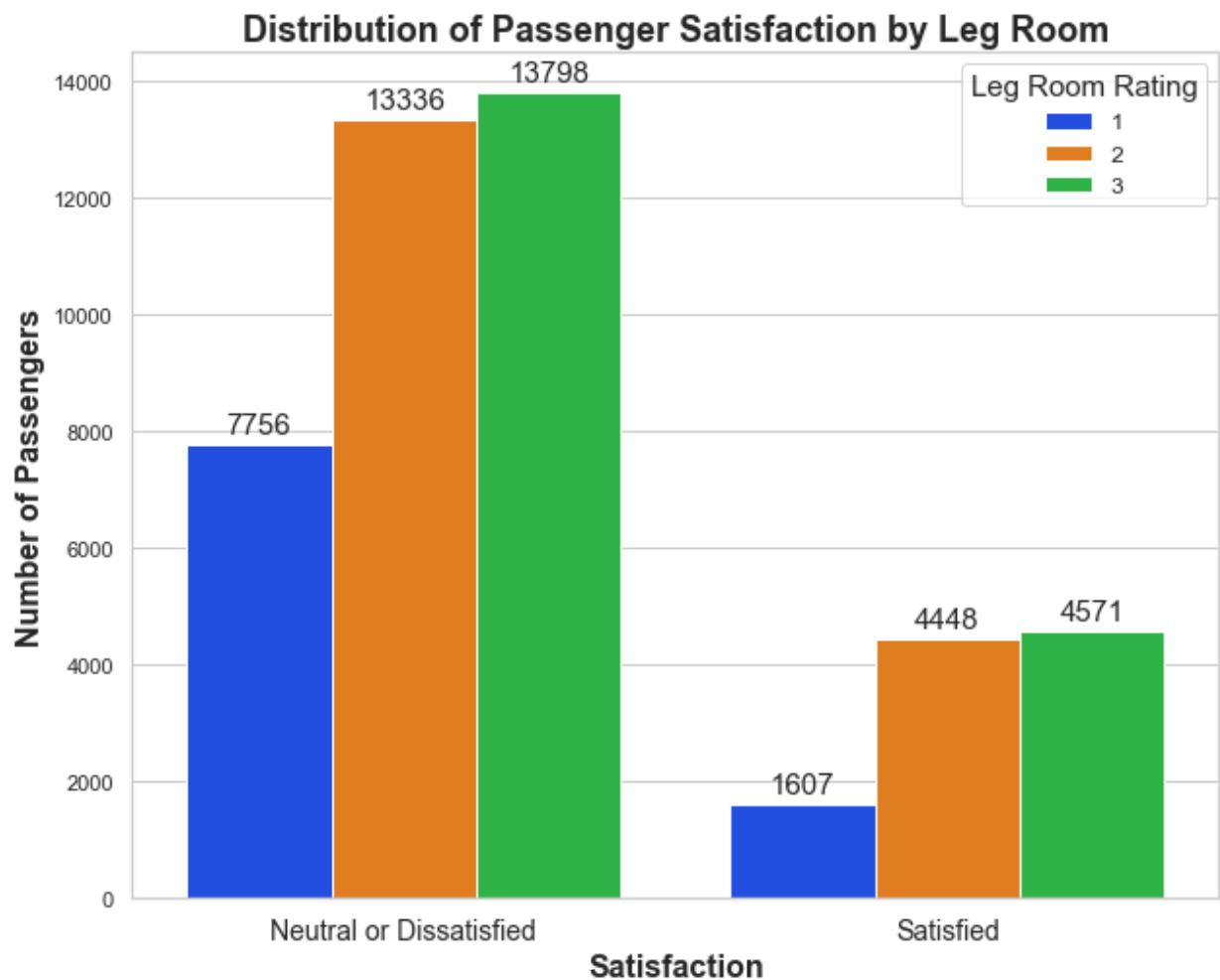
```
In [30]: 1 # Group satisfaction by leg room rating
2 leg_room_group = data.groupby(['satisfaction', 'leg_room'])['leg_room']
3 leg_room_group
```

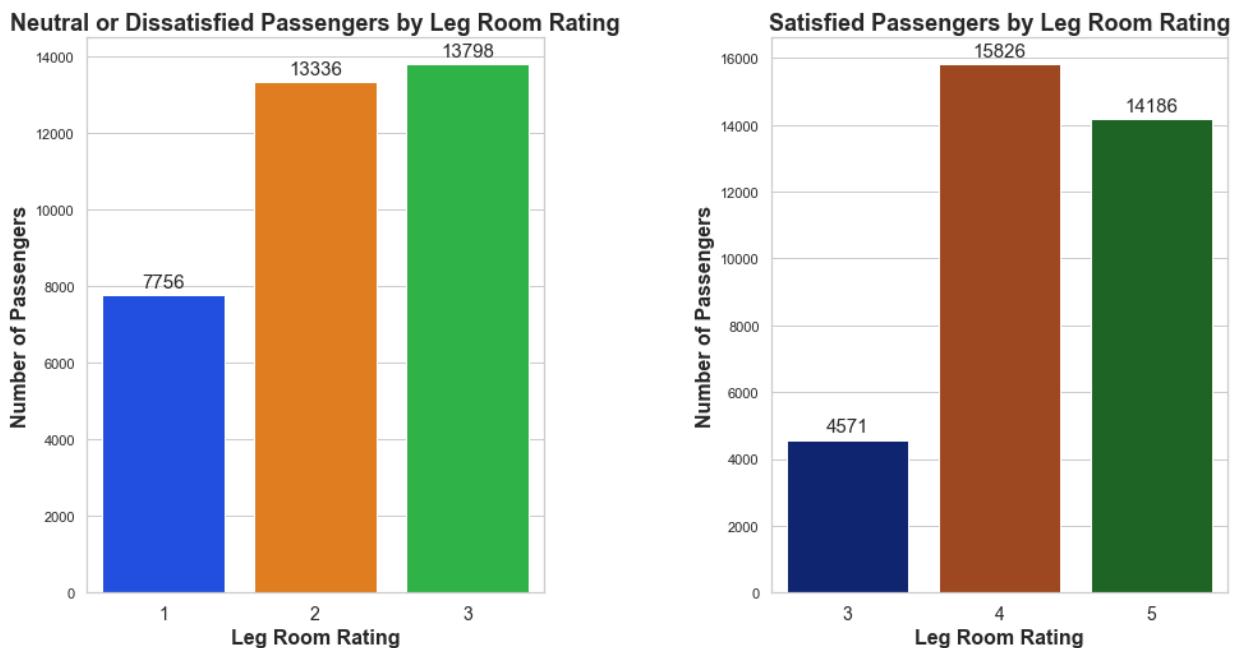
```
Out[30]: satisfaction  leg_room
0              1        7756
              2       13336
              3       13798
              4       11178
              5        8709
1              1        1607
              2        4448
              3        4571
              4       15826
              5       14186
Name: leg_room, dtype: int64
```


In [31]:

```
1 # set seaborn style to whitegrid
2 sns.set_style('whitegrid')
3 #set the plot size
4 plt.figure(figsize=(10, 8))
5
6 # Create countplot to visualize satisfaction by leg room. make the colo
7 ax = sns.countplot(x='satisfaction', hue='leg_room', data=data, palette
8 # set the title and make it bold
9 plt.title('Distribution of Passenger Satisfaction by Leg Room', fontwei
10 #set the xlabel and make it bold
11 plt.xlabel('Satisfaction', fontweight='bold', fontsize='16')
12 #set the ylabel and make it bold
13 plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')
14 # set the xticks labels
15 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=1
16
17 # Add count values to the top of each bar
18 for p in ax.patches:
19     ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width(
20             ha = 'center', va = 'center', xytext = (0, 10), textcoo
21
22 # Change the labels of the legend
23 handles, labels = ax.get_legend_handles_labels()
24 ax.legend(handles, ['1', '2', '3'], fontsize='12', title='Leg Room Rati
25
26 plt.savefig('images/leg_room.png', format='png')
27
28 # Create two separate data frames for not satisfied and satisfied group
29 not_satisfied = data.loc[data['satisfaction'] == 0]
30 satisfied = data.loc[data['satisfaction'] == 1]
31
32 # Create countplots for each group separately
33 fig, axes = plt.subplots(1, 2, figsize=(16, 8))
34
35
36 sns.set_style('whitegrid')
37
38 # Plot countplot for not satisfied group
39 sns.countplot(x='leg_room', data=not_satisfied, palette='bright', hue_o
40 axes[0].set_title('Neutral or Dissatisfied Passengers by Leg Room Rating',
41 axes[0].set_xlabel('Leg Room Rating', fontweight='bold', fontsize='16')
42 axes[0].set_ylabel('Number of Passengers', fontweight='bold', fontsize=
43 axes[0].set_xticklabels(['1', '2', '3'], fontsize=14)
44
45 # Add count values to the top of each bar
46 for p in axes[0].patches:
47     axes[0].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_w
48             ha = 'center', va = 'center', xytext = (0, 10), textcoo
49
50 # Plot countplot for satisfied group
51 sns.countplot(x='leg_room', data=satisfied, palette='dark', hue_order=[
52 axes[1].set_title('Satisfied Passengers by Leg Room Rating', fontweight
53 axes[1].set_xlabel('Leg Room Rating', fontweight='bold', fontsize='16')
54 axes[1].set_ylabel('Number of Passengers', fontweight='bold', fontsize=
55 axes[1].set_xticklabels(['3', '4', '5'], fontsize=14)
56
57 # Add count values to the top of each bar
```

```
58 for p in axes[1].patches:
59     axes[1].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_
60                             ha = 'center', va = 'center', xytext = (0, 10), textcoo
61
62 # create more space between the two bottom graphs
63 plt.subplots_adjust(wspace=0.5, hspace=0.5)
64
65 plt.savefig('images/leg_room_2.png', format='png')
66
67
68
69 plt.show()
```





Observations:

- From the graphs above, it is clear that if a airline passenger rated their leg room with a rating of 1-3, they were more likely to be dissatisfied with their experience.
- Oppositely, if a passenger had a rating of 4-5 the likelihood of them being satisfied with their experience is much higher.

Seat Comfort

- Seat comfort is a crucial factor for airline customers as it directly affects their overall satisfaction with the flight experience.
- Passengers spend hours sitting in their seats, and uncomfortable seats can lead to physical discomfort, fatigue, and even injuries such as back pain.
- Additionally, uncomfortable seats can negatively impact a passenger's mood and perception of the flight, leading to lower levels of satisfaction and potentially deterring them from flying with the airline again in the future.
- Ensuring comfortable seats can be a key differentiator for airlines in a highly competitive industry.

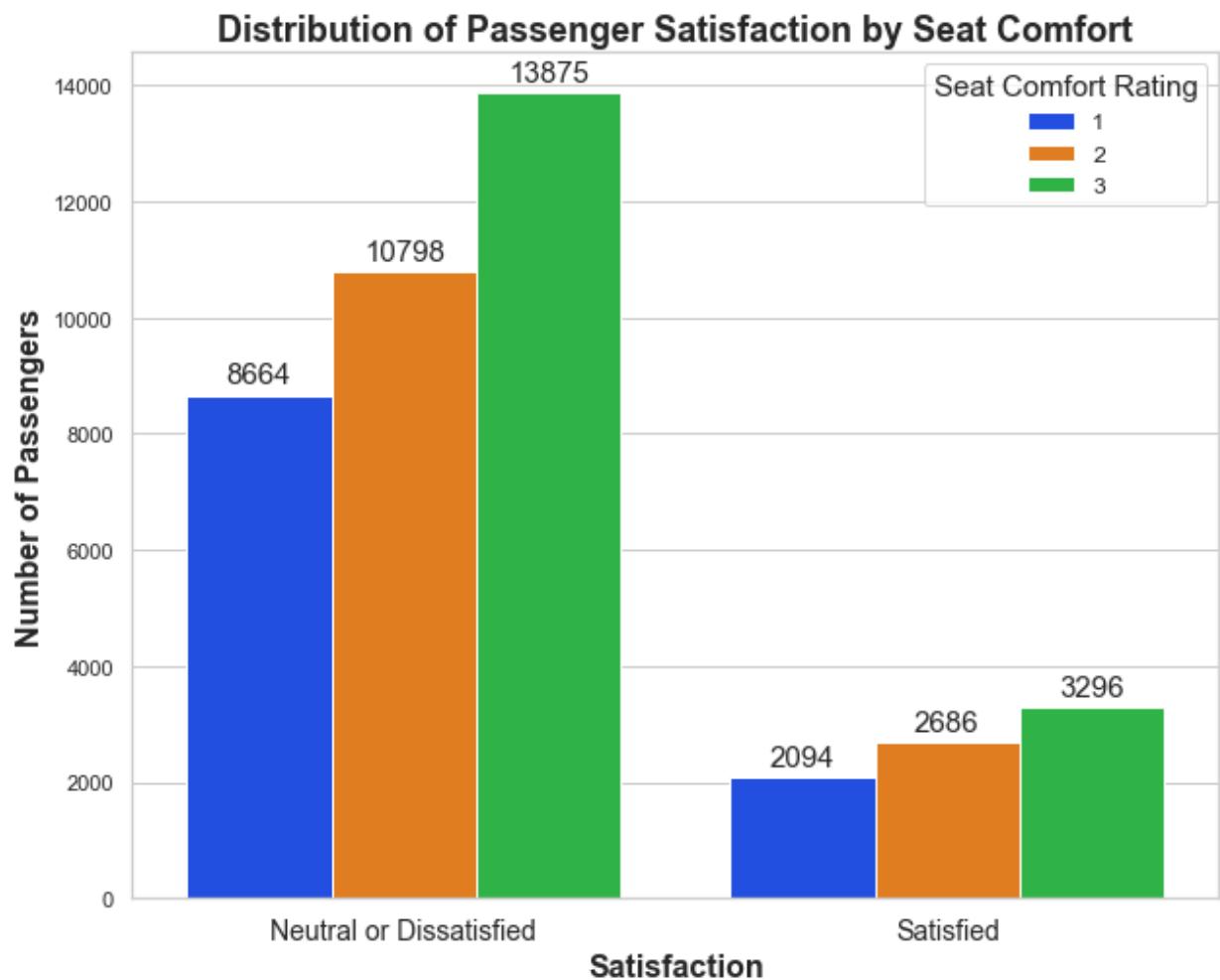
```
In [32]: 1 # group satisfaction by seat comfort rating
          2 seat_comfort_group = data.groupby(['satisfaction', 'seat_comfort'])['se
          3 seat_comfort_group
```

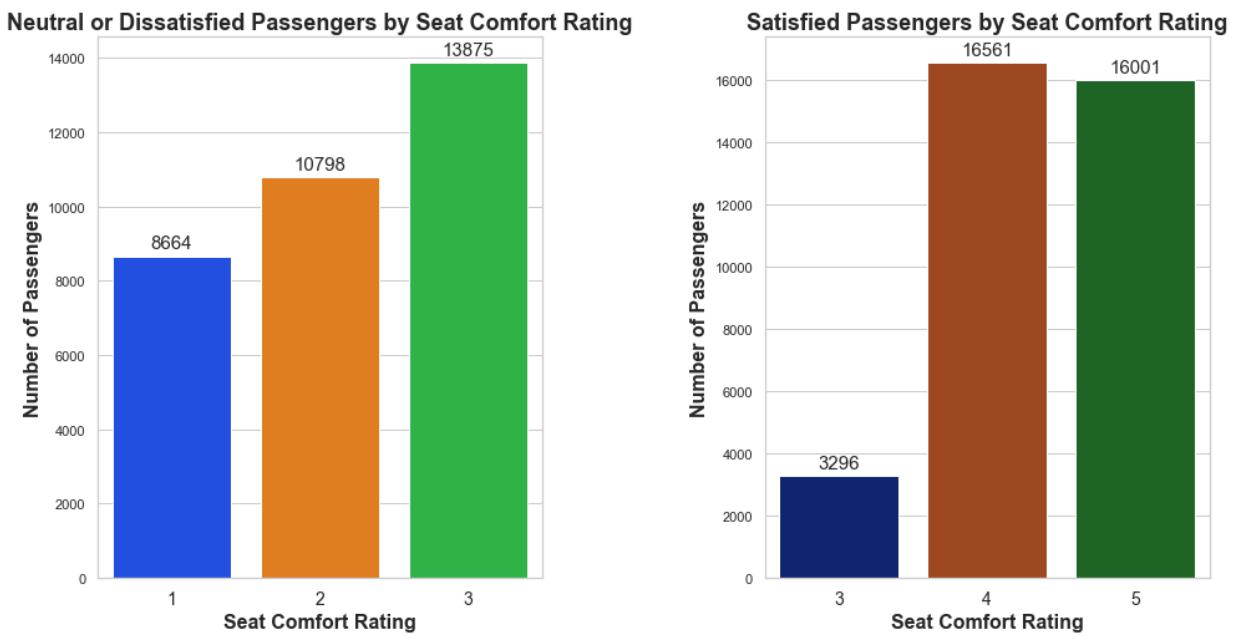
```
Out[32]: satisfaction  seat_comfort
          0             1           8664
                         2          10798
                         3          13875
                         4          13016
                         5           8424
          1             1           2094
                         2          2686
                         3          3296
                         4          16561
                         5          16001
Name: seat_comfort, dtype: int64
```


In [33]:

```
1 # Set seaborn style to whitegrid
2 sns.set_style('whitegrid')
3
4 plt.figure(figsize=(10, 8))
5
6 #create a countplot to visualize satisfaction by seat comfort. With the
7 ax = sns.countplot(x='satisfaction', hue='seat_comfort', data=data, pal
8 # Set title and make it bold
9 plt.title('Distribution of Passenger Satisfaction by Seat Comfort', fon
10 # Set xlabel and make it bold
11 plt.xlabel('Satisfaction', fontweight='bold', fontsize='16')
12 #set ylabel and make it bold
13 plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')
14 # set the xticks labels
15 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=1
16
17 # Add count values to the top of each bar
18 for p in ax.patches:
19     ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width
20             ha = 'center', va = 'center', xytext = (0, 10), textcoo
21
22 # Change the labels of the legend
23 handles, labels = ax.get_legend_handles_labels()
24 ax.legend(handles, ['1', '2', '3'], fontsize='12', title='Seat Comfort'
25
26 plt.savefig('images/seat_comfort.png', format='png')
27
28 # Create two separate data frames for not satisfied and satisfied group
29 not_satisfied = data.loc[data['satisfaction'] == 0]
30 satisfied = data.loc[data['satisfaction'] == 1]
31
32 # Create countplots for each group separately
33 fig, axes = plt.subplots(1, 2, figsize=(16, 8))
34
35
36 sns.set_style('whitegrid')
37
38 # Plot countplot for not satisfied group
39 sns.countplot(x='seat_comfort', data=not_satisfied, palette='bright', h
40 axes[0].set_title('Neutral or Dissatisfied Passengers by Seat Comfort R
41 axes[0].set_xlabel('Seat Comfort Rating', fontweight='bold', fontsize='
42 axes[0].set_ylabel('Number of Passengers', fontweight='bold', fontsize=
43 axes[0].set_xticklabels(['1', '2', '3'], fontsize=14)
44
45 # Add count values to the top of each bar
46 for p in axes[0].patches:
47     axes[0].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_
48             ha = 'center', va = 'center', xytext = (0, 10), textcoo
49
50 # Plot countplot for satisfied group
51 sns.countplot(x='seat_comfort', data=satisfied, palette='dark', hue_ord
52 axes[1].set_title('Satisfied Passengers by Seat Comfort Rating', fontwe
53 axes[1].set_xlabel('Seat Comfort Rating', fontweight='bold', fontsize='
54 axes[1].set_ylabel('Number of Passengers', fontweight='bold', fontsize=
55 axes[1].set_xticklabels(['3', '4', '5'], fontsize=14)
56
57 # Add count values to the top of each bar
```

```
58 for p in axes[1].patches:  
59     axes[1].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_  
60             ha = 'center', va = 'center', xytext = (0, 10), textcoo  
61  
62 # create more space between the two bottom graphs  
63 plt.subplots_adjust(wspace=0.5, hspace=0.5)  
64  
65 plt.savefig('images/seat_comfort_2.png', format='png')  
66  
67  
68  
69 plt.show()
```





Observations:

- It is evident, that passengers that were satisfied were consistently rating their seat comfort as a 4 or 5.
- Oppositely, passengers that were neutral/dissatisfied had a rating of 1-3.
- In the top graph, it is clear that if a passenger rated their seat comfort from 1-3, they greatly increased the chance to be neutral or dissatisfied.
- I believe comfortable seats are a feature that a lot of airlines overlook. Having comfortable seats could be a huge difference maker in a competitive market.

Inflight Service

- Inflight service is important for airlines because it plays a significant role in enhancing the overall experience of passengers during a flight.
- This includes factors such as food and drink service, onboard entertainment, on-board service, leg room service, and cleanliness.
- Inflight service can help differentiate an airline from its competitors and can influence a passenger's decision to choose one airline over another.
- Additionally, providing good inflight service can lead to higher customer satisfaction and increased loyalty, which can lead to repeat business and positive word-of-mouth recommendations.
- Therefore, airlines should place a strong emphasis on providing high-quality inflight service to their passengers.

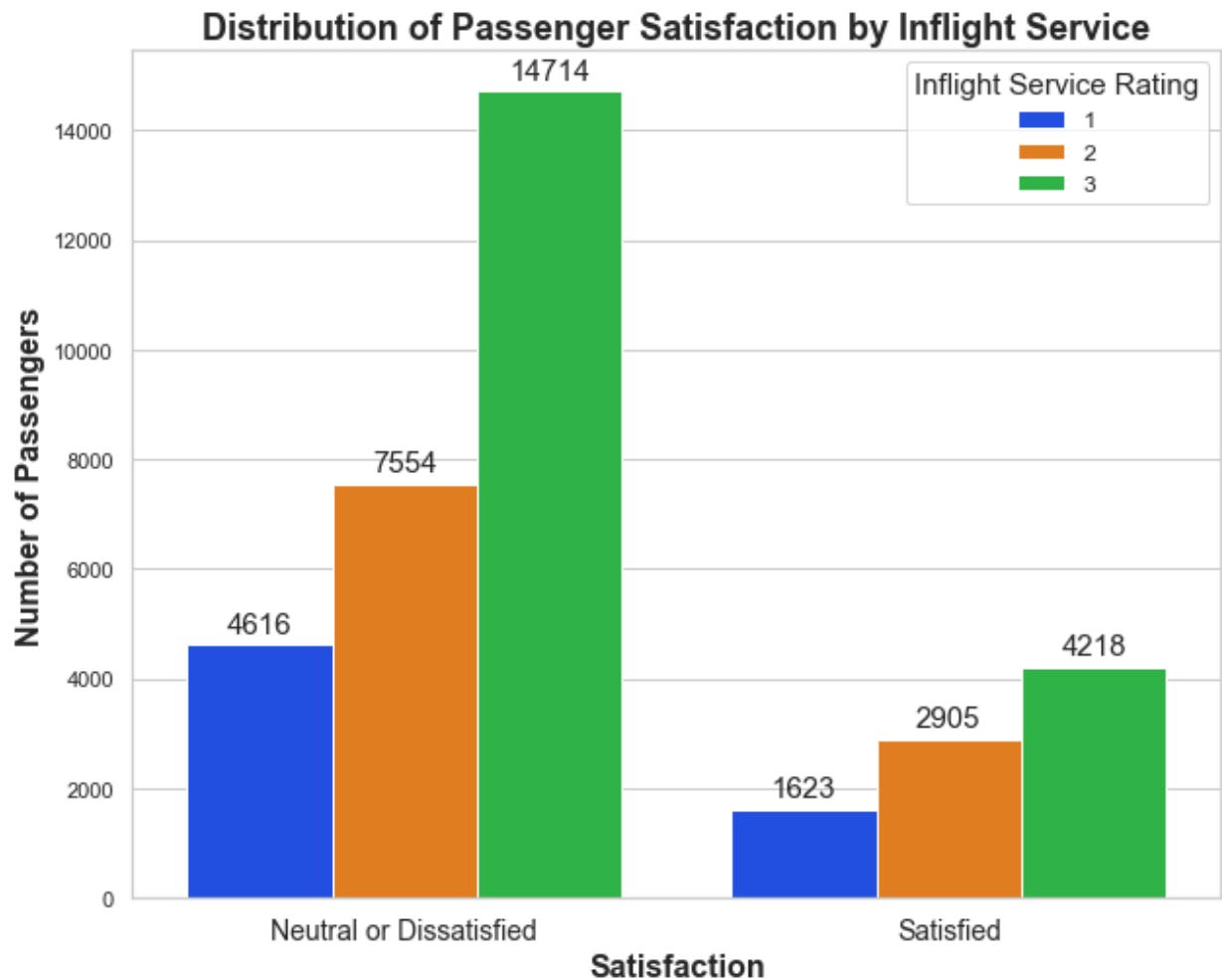
```
In [34]: 1 inflight_service_group = data.groupby(['satisfaction', 'inflight_service'])
2 inflight_service_group
```

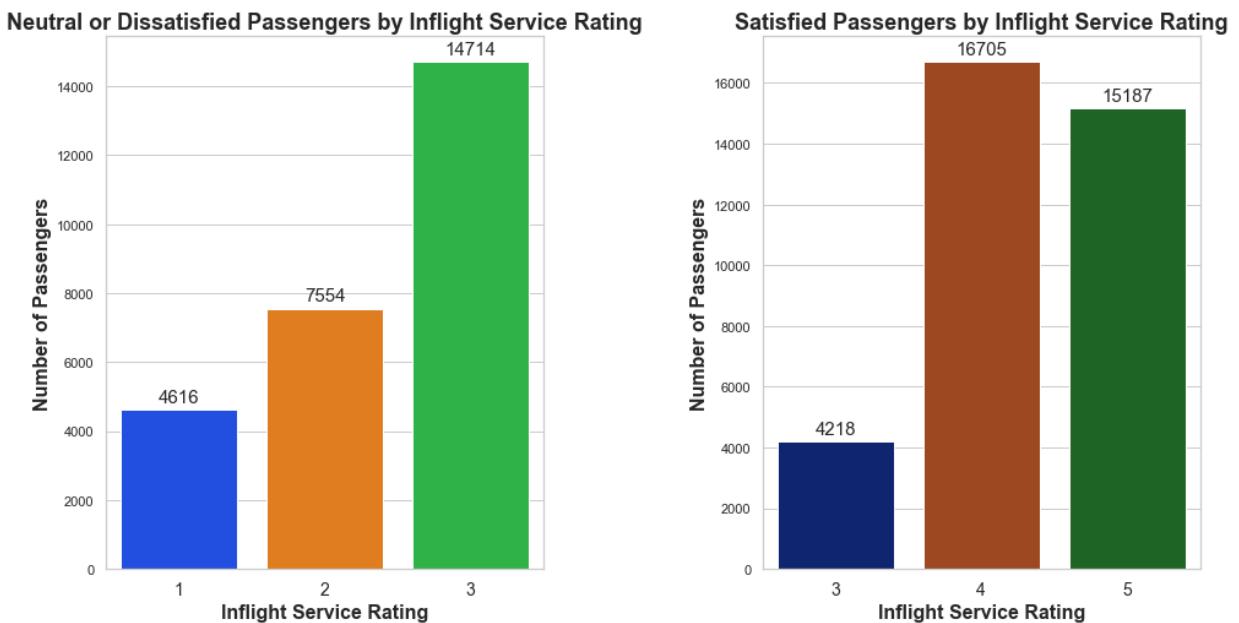
```
Out[34]: satisfaction  inflight_service
          0             1                  4616
                         2                  7554
                         3                 14714
                         4                 18427
                         5                  9466
          1             1                  1623
                         2                  2905
                         3                  4218
                         4                 16705
                         5                 15187
Name: inflight_service, dtype: int64
```


In [35]:

```
1 # Set seaborn style to white grid
2 sns.set_style('whitegrid')
3
4 plt.figure(figsize=(10, 8))
5 # create a countplot to vizualize satisfaction by inflight service. make
6 ax = sns.countplot(x='satisfaction', hue='inflight_service', data=data,
7 # set the title and make it bold
8 plt.title('Distribution of Passenger Satisfaction by Inflight Service',
9 #set the x label and make it bold
10 plt.xlabel('Satisfaction', fontweight='bold', fontsize='16')
11 #set the y label and make it bold
12 plt.ylabel('Number of Passengers', fontweight='bold', fontsize='16')
13 #set the xticks labels
14 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=1
15
16 # Add count values to the top of each bar
17 for p in ax.patches:
18     ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width
19             ha = 'center', va = 'center', xytext = (0, 10), textcoo
20
21 # Change the labels of the legend
22 handles, labels = ax.get_legend_handles_labels()
23 ax.legend(handles, ['1', '2', '3'], fontsize='12', title='Inflight Serv
24
25 plt.savefig('images/inflight_service.png', format='png')
26
27 # Create two separate data frames for not satisfied and satisfied group
28 not_satisfied = data.loc[data['satisfaction'] == 0]
29 satisfied = data.loc[data['satisfaction'] == 1]
30
31 # Create countplots for each group separately
32 fig, axes = plt.subplots(1, 2, figsize=(16, 8))
33
34
35 sns.set_style('whitegrid')
36
37 # Plot countplot for not satisfied group
38 sns.countplot(x='inflight_service', data=not_satisfied, palette='bright
39 axes[0].set_title('Neutral or Dissatisfied Passengers by Inflight Servi
40 axes[0].set_xlabel('Inflight Service Rating', fontweight='bold', fontsi
41 axes[0].set_ylabel('Number of Passengers', fontweight='bold', fontsize=
42 axes[0].set_xticklabels(['1', '2', '3'], fontsize=14)
43
44 # Add count values to the top of each bar
45 for p in axes[0].patches:
46     axes[0].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_
47             ha = 'center', va = 'center', xytext = (0, 10), textcoo
48
49 # Plot countplot for satisfied group
50 sns.countplot(x='inflight_service', data=satisfied, palette='dark', hue
51 axes[1].set_title('Satisfied Passengers by Inflight Service Rating', fo
52 axes[1].set_xlabel('Inflight Service Rating', fontweight='bold', fontsi
53 axes[1].set_ylabel('Number of Passengers', fontweight='bold', fontsize=
54 axes[1].set_xticklabels(['3', '4', '5'], fontsize=14)
55
56 # Add count values to the top of each bar
57 for p in axes[1].patches:
```

```
58     axes[1].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_
59                     ha = 'center', va = 'center', xytext = (0, 10), textcoo
60
61 # create more space between the two bottom graphs
62 plt.subplots_adjust(wspace=0.5, hspace=0.5)
63
64 plt.savefig('images/inflight_service_2.png', format='png')
65
66 plt.show()
```





Observations:

- It is evident that great inflight services well result in satisfied customers.
- Oppositely, if the inflight services are getting rated 1-3, it highly increases the chances that the passengers are going to be dissatisfied.
- If the passengers rated the inflight service with 4 or 5 ratings, it greatly increases the likelihood of the passengers being satisfied.

Inflight Entertainment

- Some could argue that inflight entertainment is equally as important as any while flying. It truly enhances the overall passenger experience during the flight.
- By providing a variety of entertainment options such as movies, TV shows, music, games, and more, airlines can help to alleviate boredom and make long flights more enjoyable for passengers. This can be especially important for longer flights.

```
In [36]: 1 #group satisfaction by inflight entertainment rating
           2 entertainment_group = data.groupby(['satisfaction', 'inflight_entertainment'])
           3 entertainment_group
```

Out[36]: satisfaction inflight_entertainment

satisfaction	inflight_entertainment	Count
0	1	10014
0	2	13086
0	3	13095
0	4	10542
0	5	8040
1	1	1050
1	2	2867
1	3	4236
1	4	17085
1	5	15400

Name: inflight_entertainment, dtype: int64

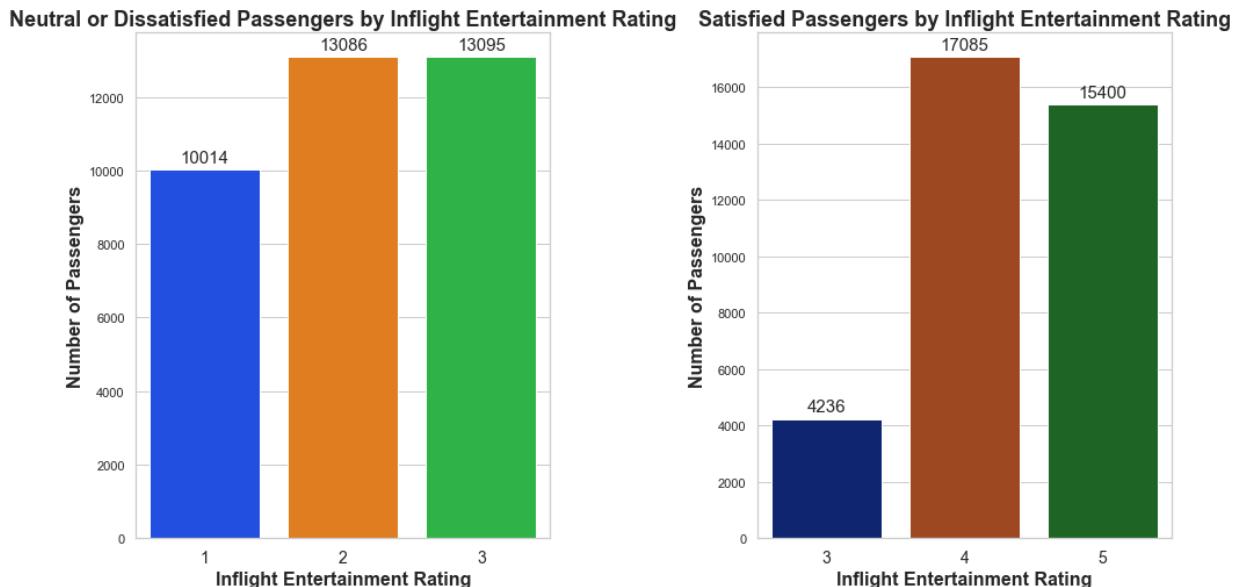
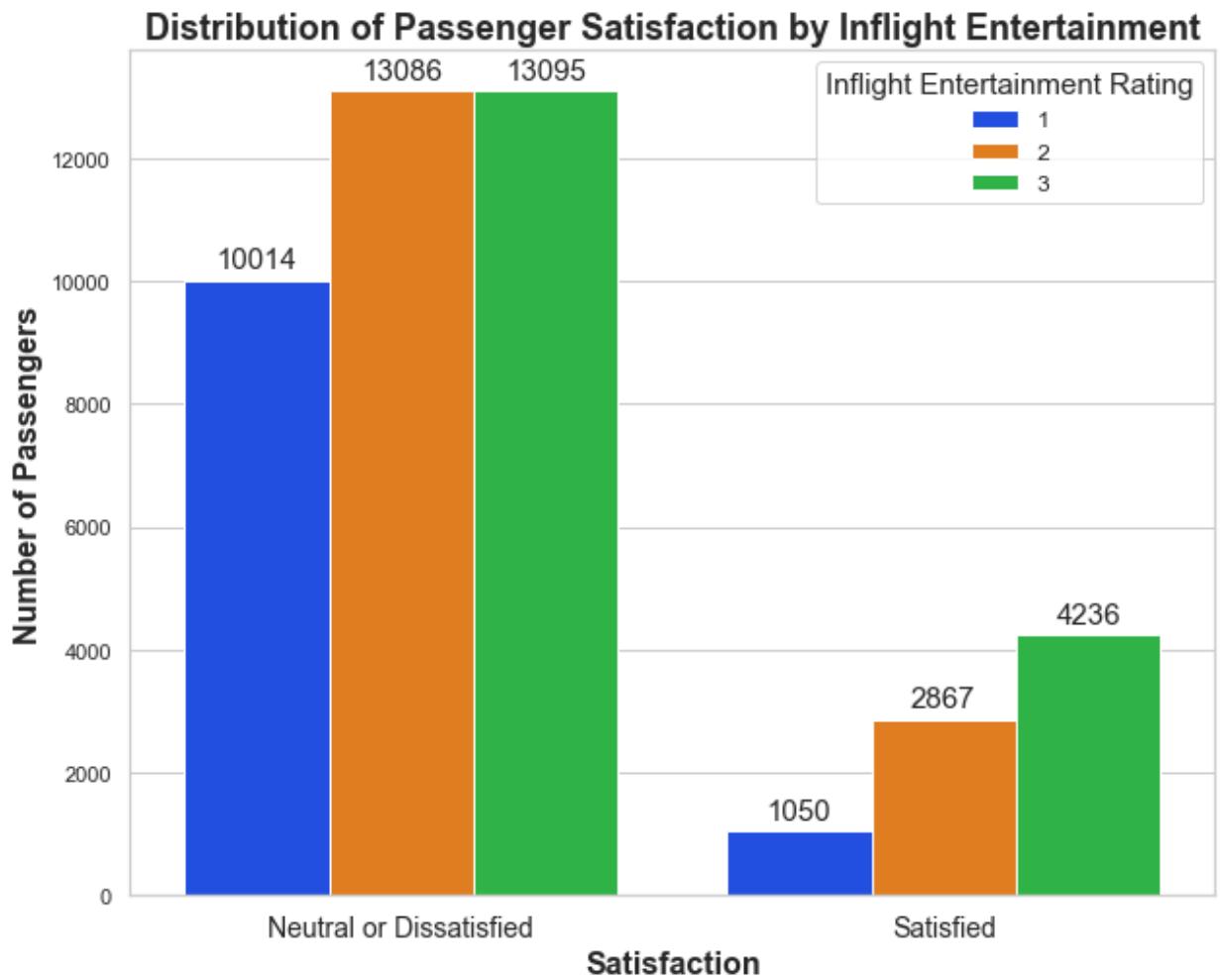
In [37]:

```
1 # set seaborn style to whitegrid
2 sns.set_style('whitegrid')
3
4 plt.figure(figsize=(10, 8))
5
6 # Create a countplot to visualize satisfaction by inflight entertainment
7 ax = sns.countplot(x='satisfaction', hue='inflight_entertainment', data=
8
9 # Set the title, x-axis label, and y-axis label of the plot.
10 plt.title('Distribution of Passenger Satisfaction by Inflight Entertainment', fontweight='bold', fontsize=16)
11 plt.xlabel('Satisfaction', fontweight='bold', fontsize=16)
12 plt.ylabel('Number of Passengers', fontweight='bold', fontsize=16)
13 ax.set_xticklabels(['Neutral or Dissatisfied', 'Satisfied'], fontsize=14)
14
15 # Add count values to the top of each bar
16 for p in ax.patches:
17     ax.annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2, p.get_y() + p.get_height() / 2), ha = 'center', va = 'center', xytext = (0, 10), textcolor='white')
18
19
20 # Change the labels of the legend
21 handles, labels = ax.get_legend_handles_labels()
22 ax.legend(handles, ['1', '2', '3'], fontsize=12, title='Inflight Entertainment Rating')
23
24 plt.savefig('images/entertain.png', format='png')
25
26 # Create two separate data frames for not satisfied and satisfied group
27 not_satisfied = data.loc[data['satisfaction'] == 0]
28 satisfied = data.loc[data['satisfaction'] == 1]
29
30 # Create countplots for each group separately
31 fig, axes = plt.subplots(1, 2, figsize=(16, 8))
32
33
34 sns.set_style('whitegrid')
35
36 # Plot countplot for not satisfied group
37 sns.countplot(x='inflight_entertainment', data=not_satisfied, palette='dark'
38 axes[0].set_title('Neutral or Dissatisfied Passengers by Inflight Entertainment Rating', fontweight='bold', fontsize=16)
39 axes[0].set_xlabel('Inflight Entertainment Rating', fontweight='bold', fontsize=14)
40 axes[0].set_ylabel('Number of Passengers', fontweight='bold', fontsize=14)
41 axes[0].set_xticklabels(['1', '2', '3'], fontsize=14)
42
43 # Add count values to the top of each bar
44 for p in axes[0].patches:
45     axes[0].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2, p.get_y() + p.get_height() / 2), ha = 'center', va = 'center', xytext = (0, 10), textcolor='white')
46
47
48 # Plot countplot for satisfied group
49 sns.countplot(x='inflight_entertainment', data=satisfied, palette='dark'
50 axes[1].set_title('Satisfied Passengers by Inflight Entertainment Rating', fontweight='bold', fontsize=16)
51 axes[1].set_xlabel('Inflight Entertainment Rating', fontweight='bold', fontsize=14)
52 axes[1].set_ylabel('Number of Passengers', fontweight='bold', fontsize=14)
53 axes[1].set_xticklabels(['3', '4', '5'], fontsize=14)
54
55 # Add count values to the top of each bar
56 for p in axes[1].patches:
57     axes[1].annotate(format(p.get_height(), '.0f'), (p.get_x() + p.get_width() / 2, p.get_y() + p.get_height() / 2), ha = 'center', va = 'center', xytext = (0, 10), textcolor='white')
```

```

58
59
60 #create space between the two bottom graphs
61 plt.subplots_adjust(wspace=0.5, hspace=0.5)
62
63 plt.savefig('images/entertain_2.png', format='png')
64
65 plt.show()

```



Observations:

- The Above graphs show that when passengers rate their inflight entertainment from 1-3, it is a strong likelihood that they will be classified as neutral or dissatisfied.
- On the other hand, when the rating for inflight entertainment was a 4 or 5, the likelihood of the passenger being classified as satisfied was very high.
- It is evident, that inflight entertainment is a strong factor when trying to give airline passengers a great experience.

Feature Selection

- Feature selection is an important process for several reasons:
 - Improves model performance.
 - You want to remove irrelevant features.
 - Reduces overfitting.
 - Too many features can lead to overfitting.
 - Reducing computational time and storage requirements.
 - Improving interpretability.
 - Using fewer features can make the model easier to understand.

```
In [38]: 1 # print first 5 rows of the dataset
           2 data.head()
```

Out[38]:

	gender	customer_type	age	type_of_travel	class	flight_distance	inflight_wifi_service	departu
0	Male	Returning Customer	13	Personal Travel	Eco Plus	460		3
1	Male	First-time Customer	25	Business travel	Business	235		3
2	Female	Returning Customer	26	Business travel	Business	1142		2
3	Female	Returning Customer	25	Business travel	Business	562		2
4	Male	Returning Customer	61	Business travel	Business	214		3

Correlation Matrix and Heatmap

- This is the next step of feature selection.
- When reading a heat map and correlation matrix, you want to look at each coordinate and select the largest numbers.
 - The closer the number is to 1, the higher the correlation, making it a strong feature to select for your modeling.
- The heat map is a more visually appealing version of a correlation matrix. The lighter the color is, the weaker the relationship is.

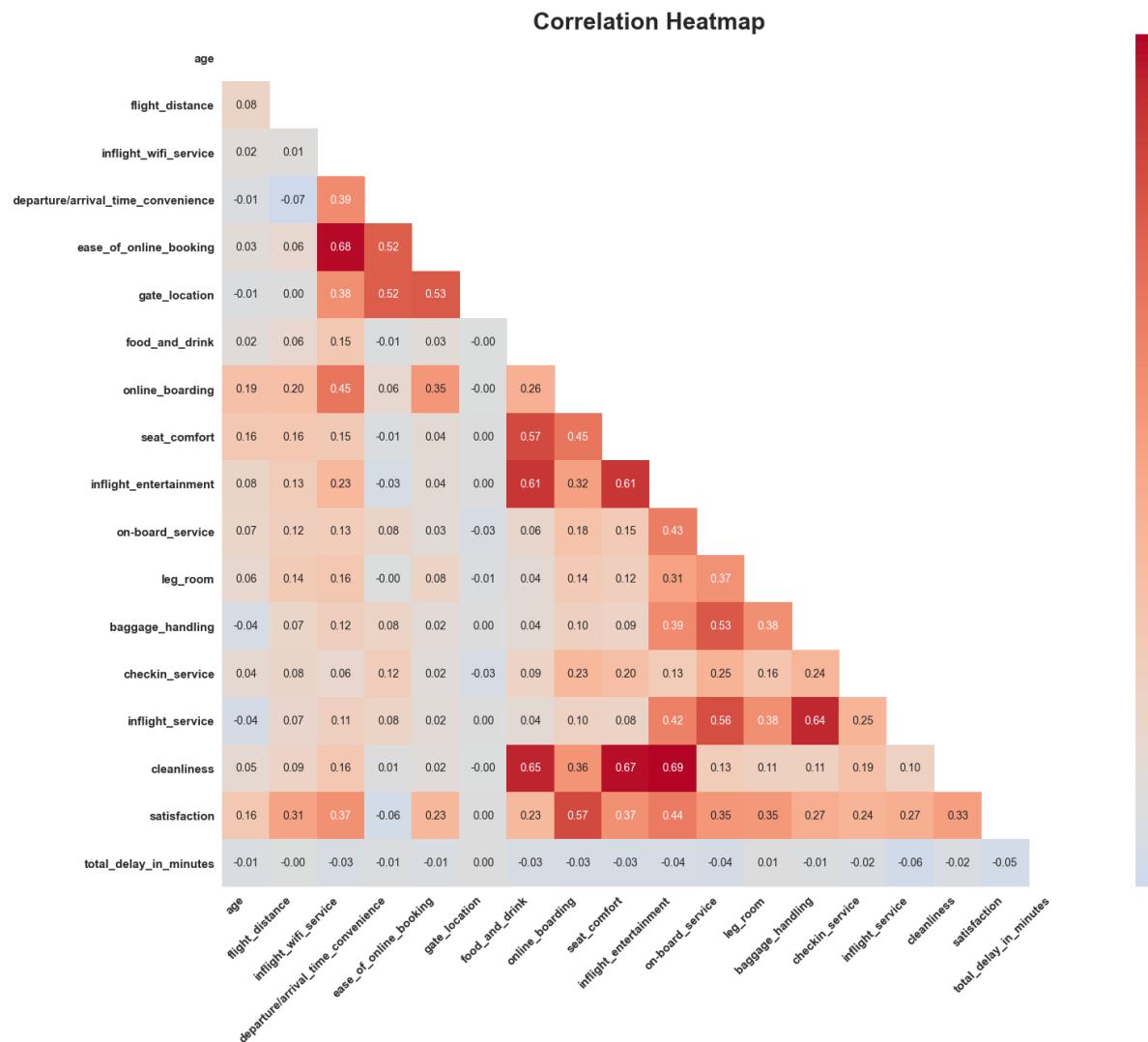
- In this case I visually detect 5 variables that have weak relationships with our target variable, 'satisfaction'.
 - **age**
 - **departure/arrival_time_convenience**
 - **gate_location**
 - **total_delay_in_minutes**

In [39]:

```

1 # Create a heatmap using Seaborn's 'sns.heatmap()' function to visualize
2 # The correlation matrix is first calculated using the 'corr()' method
3 # The heatmap is plotted using the 'corr_matrix', with the mask parameter
4 # The 'annot' parameter is set to True to display the correlation value
5 # The 'cmap' parameter is used to specify the color scheme for the plot
6 # The 'center' parameter sets the value at which the colors will be centered
7 corr_matrix = data.corr()
8 fig, ax = plt.subplots(figsize=(27,20))
9 mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
10 sns.heatmap(corr_matrix, mask=mask, annot=True, annot_kws={"fontsize": 11}
11
12 # Format the x-axis and y-axis labels, as well as the title of the plot
13 ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right', fontsize=15, fontweight='bold')
14 ax.set_yticklabels(ax.get_yticklabels(), fontsize=15, fontweight='bold')
15 plt.title('Correlation Heatmap', fontweight='bold', fontsize = 30)
16
17 plt.show()

```



```
In [40]: 1 # correlation matrix as satisfaction as the target
2 satisfaction_corr = data.corr()['satisfaction'].map(abs).sort_values(as
3 satisfaction_corr
```

Out[40]:

satisfaction	1.000000
online_boarding	0.569430
inflight_entertainment	0.438129
seat_comfort	0.374924
inflight_wifi_service	0.374004
on-board_service	0.349038
leg_room	0.345042
cleanliness	0.327679
flight_distance	0.307627
baggage_handling	0.272755
inflight_service	0.268044
checkin_service	0.242945
ease_of_online_booking	0.230243
food_and_drink	0.226694
age	0.155291
departure/arrival_time_convenience	0.056881
total_delay_in_minutes	0.054974
gate_location	0.004567

Name: satisfaction, dtype: float64

- Just to confirm with the heatmap, this list of variables confirms that in fact age, departure/arrival_time_convenience, total_delay_in_minutes, and gate_location have the weakest relationships to our target variable satisfaction.
- The next step will be to remove these variables from the dataset before we begin modeling.

Drop weak correlated variables

- Improving model accuracy: Weakly correlated variables are not providing much useful information to the model, so removing them can improve the accuracy of the model by reducing noise and allowing the model to focus on the most relevant variables.
- Reducing complexity: Including too many variables in a model can lead to overfitting and decrease the model's generalizability. Removing weakly correlated variables reduces the complexity of the model and can prevent overfitting.
- Speeding up computation: Including too many variables in a model can also increase the computational cost and slow down the training and prediction time. Removing weakly correlated variables can speed up computation and make the model more efficient.

```
In [41]: 1 # drop columns
2 data.drop(['age', 'departure/arrival_time_convenience', 'total_delay_in
3 'gate_location'], axis=1, inplace=True)
```

In [42]:

```

1 # Check if columns have been removed from the final dataframe
2 print(data.columns)

Index(['gender', 'customer_type', 'type_of_travel', 'class', 'flight_distance',
       'inflight_wifi_service', 'ease_of_online_booking', 'food_and_drink',
       'online_boarding', 'seat_comfort', 'inflight_entertainment',
       'on-board_service', 'leg_room', 'baggage_handling', 'checkin_service',
       'inflight_service', 'cleanliness', 'satisfaction'],
      dtype='object')

```

- age, departure/arrival_time_convenience, total_delay_in_minutes, and gate_location have been dropped from the dataset

Prepare Data for Modeling

- Create X and Y variables.
 - X = independent variables.
 - y = dependent variable.
- Split the data in 80/20 train/test.

Create X and y variables

In [43]:

```

1 # Create X
2 X = data.drop('satisfaction', axis=1)
3
4 # Create y
5 y = data['satisfaction']

```

Perform train/test split

- The original dataset is divided into two parts: X, which contains the features, and y, which contains the target variable (in this case, the passenger satisfaction).
- The function train_test_split is used to randomly split the data into a training set (X_train, y_train) and a test set (X_test, y_test). The parameter test_size is set to 0.25, which means that 25% of the data will be used for testing and the remaining 75% for training.
- The parameter random_state is set to 42 to ensure that the same random splits are generated every time the code is run. This is important for reproducibility and to ensure that the results can be compared across different runs.

In [44]:

```

1 # create a train test split with a 75/25 split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

```

Baseline Model : Logistic Regression

Define Which Columns To Encode

- This block of code show the how the categorical data is going to be divided up prior to encoding in the pipeline.
- Ordinal categorical data is the survey columns that have rating from 1-5.
- Categorical data (gender, customer_type, type_of_travel, and class) are nominal because there is no order or hierarchy.

```
In [45]: 1 # create variable for ordinal categorical data
2 ordinal_cols = ['inflight_wifi_service',
3                  'ease_of_online_booking',
4                  'food_and_drink',
5                  'online_boarding',
6                  'seat_comfort',
7                  'inflight_entertainment',
8                  'on-board_service',
9                  'leg_room',
10                 'baggage_handling',
11                 'checkin_service',
12                 'inflight_service',
13                 'cleanliness']
14
15 # create categorical_cols that are nominal
16 categorical_cols = ['gender', 'customer_type', 'type_of_travel', 'class'
17
18 # numerical columns
19 numerical_cols = ['flight_distance']
```

Logistic Regression Pipeline

- A pipeline is a way of chaining data processing components together in a sequence, such that the output of each component is the input to the next. It allows us to streamline the workflow of data preprocessing and modeling by combining several steps into one cohesive unit.
- By using a pipeline, we can simplify the code for our machine learning tasks and reduce the risk of errors in data preprocessing, as well as making it easier to apply the same sequence of transformations to new data.

In [46]:

```
1 # Create the encoding transformer to preprocess the data
2 encoder = ColumnTransformer([
3     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len
4     ('onehot_encoder', OneHotEncoder(), categorical_cols) # encode cate
5 ], remainder='passthrough') # leave other columns unchanged
6
7 # Create the pipeline for Logistic Regression
8 pipeline = Pipeline([
9     ('encoder', encoder), # encode and preprocess data
10    ('scaler', MinMaxScaler()), # apply feature scaling to normalized b
11    ('clf', LogisticRegression(solver='liblinear', random_state=42)) #
12 ])
13
14 # Fit the pipeline on the training data
15 pipeline.fit(X_train, y_train)
16
17 #Generate predictions for the training and test data
18 y_train_pred = pipeline.predict(X_train)
19 y_test_pred = pipeline.predict(X_test)
20
21 # Print the classification reports for the training data
22 print('Classification report for logistic regression training data:')
23 print(classification_report(y_train, y_train_pred, target_names=['neutral',
24
25 # Print the confusion matrix for the test data
26 conf_mat = confusion_matrix(y_train, y_train_pred)
27 print('Confusion matrix for training data:')
28 print(conf_mat)
29
30 # Check the AUC of predictions for training data
31 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,
32 roc_auc = auc(false_positive_rate, true_positive_rate)
33 print('AUC:', roc_auc)
34 print('\n')
35 print('=====')
36
37 # Print the classification reports for the test data
38 print('Classification report for logistic regression test data:')
39 print(classification_report(y_test, y_test_pred, target_names=['neutral',
40
41 # Print the confusion matrix for the test data
42 conf_mat = confusion_matrix(y_test, y_test_pred)
43 print('Confusion matrix for test data:')
44 print(conf_mat)
45
46 # Check the AUC of predictions for test data
47 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
48 roc_auc = auc(false_positive_rate, true_positive_rate)
49 print('AUC:', roc_auc)
```

```
Classification report for logistic regression training data:
      precision    recall   f1-score   support
neutral or dissatisfied      0.91      0.91      0.91      40987
           satisfied      0.88      0.87      0.88      30574

accuracy                      0.90      0.90      0.90      71561
macro avg                     0.89      0.89      0.89      71561
weighted avg                  0.90      0.90      0.90      71561
```

Confusion matrix for training data:

```
[[37448 3539]
 [ 3924 26650]]
AUC: 0.8926556022261638
```

```
=====
Classification report for logistic regression test data:
      precision    recall   f1-score   support
neutral or dissatisfied      0.91      0.91      0.91      13790
           satisfied      0.88      0.87      0.87      10064

accuracy                      0.89      0.89      0.89      23854
macro avg                     0.89      0.89      0.89      23854
weighted avg                  0.89      0.89      0.89      23854
```

Confusion matrix for test data:

```
[[12551 1239]
 [ 1295 8769]]
AUC: 0.890737906837862
```

Observations:

- From the classification reports and confusion matrices, it can be seen that the logistic regression model performed well in predicting customer satisfaction. The precision and recall for both the training and test data are high, indicating that the model is good at correctly predicting both classes. The f1-scores are also high, indicating a good balance between precision and recall.
- The confusion matrices show that the model had a higher number of false positives for the satisfied class than false negatives, meaning that the model was more likely to incorrectly predict a satisfied customer than a dissatisfied one. However, overall, the model was able to correctly predict customer satisfaction with an accuracy of around 89%.
- The AUC score of around 0.89 indicates that the model has good discriminatory power in distinguishing between the two classes.
- Overall, the logistic regression model can be considered a good fit for the data and can be used for predicting customer satisfaction for airline passengers.

Results on Training Data

- Precision Score:** The precision for the "neutral or dissatisfied" class is 0.91, which means that 91% of the predictions for this class were correct.

- **Recall Score:** The recall for this class is 0.91, which means that 91% of the actual "neutral or dissatisfied" samples were correctly identified.
- **F1-Score:** The F1-score is the harmonic mean of precision and recall, and it is 0.91 for this class.
- **Accuracy Score:** Accuracy is the proportion of correctly classified instances. On the training data, this model has an accuracy score of 0.90.

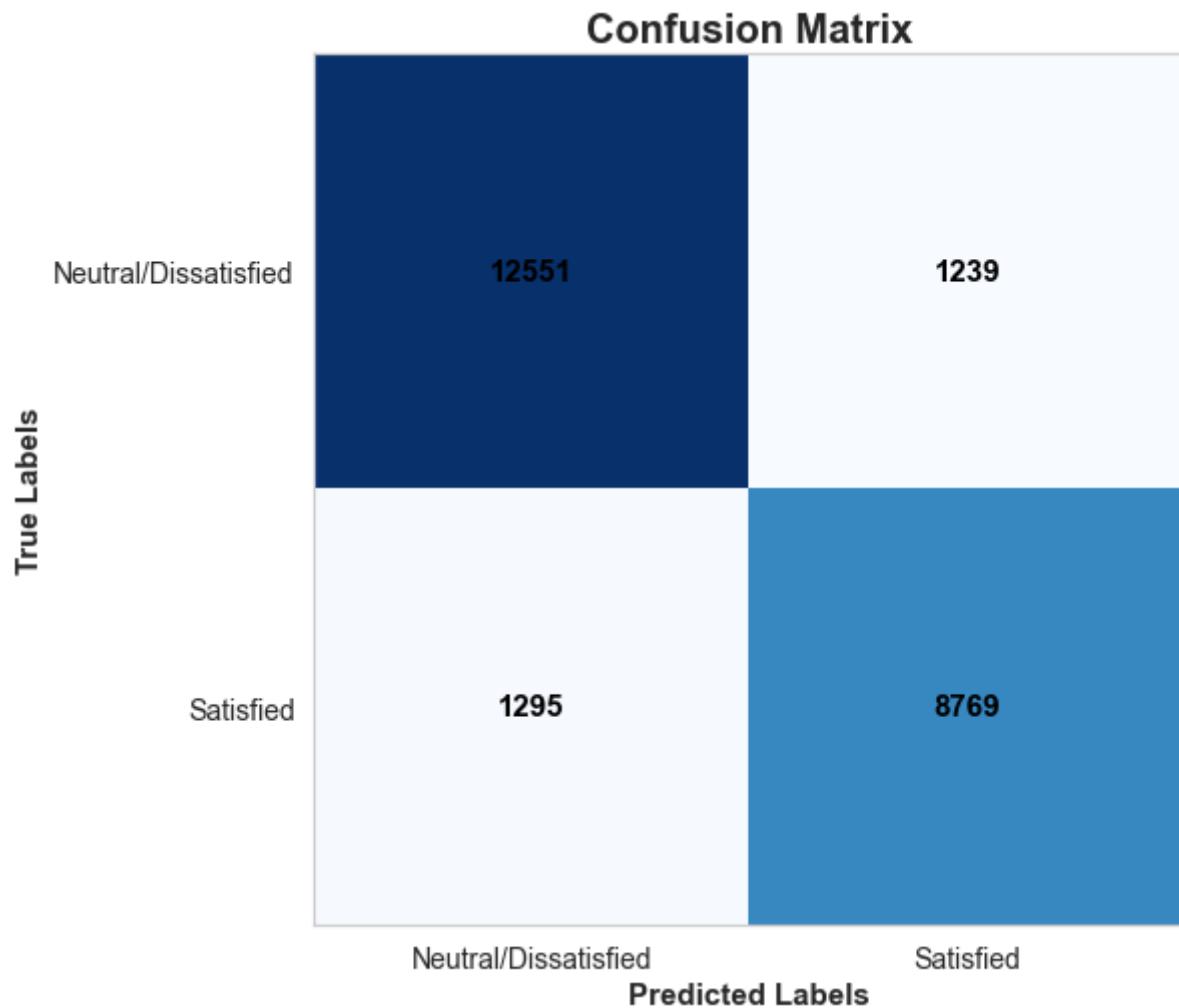
Results On Test Data:

- **Precision Score:** The precision of the model for neutral or dissatisfied customers is 91%, which means that out of all the customers predicted as neutral or dissatisfied, 91% were actually neutral or dissatisfied.
- **Recall Score:** The recall for neutral or dissatisfied customers is also 91%, which means that out of all the actual neutral or dissatisfied customers, 91% were correctly predicted as neutral or dissatisfied.
- **F1-Score:** The F1-score for neutral or dissatisfied customers is 91%.
- **Accuracy:** Overall accuracy of 89%, which means that 89% of the predictions made by the model on the test set were correct

Confusion Matrix For Test Data

In [47]:

```
1 # get confusion matrix
2 cm_lg = confusion_matrix(y_test, y_test_pred)
3
4 # plot confusion matrix
5 fig, ax = plt.subplots(figsize=(8, 8))
6 ax.imshow(cm_lg, cmap='Blues')
7 ax.grid(False)
8 ax.set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
9 ax.set_ylabel('True Labels', fontweight='bold', fontsize='15')
10 ax.set_title('Confusion Matrix', fontweight='bold', fontsize='20')
11 ax.set_xticks([0, 1])
12 ax.set_yticks([0, 1])
13 ax.xaxis.set(ticks=[0, 1], ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
14 ax.yaxis.set(ticks=[0, 1], ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
15 ax.xaxis.set_tick_params(labelsize=14)
16 ax.yaxis.set_tick_params(labelsize=14)
17
18 for i in range(2):
19     for j in range(2):
20         ax.text(j, i, cm_lg[i, j], ha='center', va='center', color='black')
```



Confusion Matrix Results:

- The confusion matrix shows that out of 13,790 neutral or dissatisfied customers in the test set, 12,551 were correctly predicted as neutral or dissatisfied (True Negative), while 1,239 were incorrectly predicted as satisfied (False Positive).
- Similarly, out of 10,064 satisfied customers in the test set, 8,769 were correctly predicted as satisfied (True Positive), while 1,295 were incorrectly predicted as neutral or dissatisfied (False

Check for Imbalance

- The data is pretty well balanced. There is no need to try SMOTE to deal with any issues with imbalanced data.

```
In [48]: 1 # check for imbalance of the training data
2 print(y_train.value_counts())
3 print('\n')
4 print(y_test.value_counts())
```

```
0    40987
1    30574
Name: satisfaction, dtype: int64
```

```
0    13790
1    10064
Name: satisfaction, dtype: int64
```

Logistic Regression Results:

- Using F1 score as the evaluation metric ensures that the model is performing well in both identifying satisfied and dissatisfied customers, which is important for predicting customer satisfaction for airline passengers.
- This model performed slightly better at identifying neutral or dissatisfied customers compared to satisfied customers.
 - **Test Data F1-Score For Satisfied Class:** 88%
 - **Test Data F1-Score For Neutral/Dissatisfied Class:** 91%
 - **AUC Score :** 89%

Model 2: Decision Tree (No Tuning)

- The first decision tree model will be a "vanilla" one, with no tuning.
- The second decision tree model will be run with GridSearchCV to find the best hyperparameters.
 - Decision Trees require a lot of pruning to be accurate. GridSearchCV is an excellent means of searching for the best hyperparameters to get the most accurate model.

In [49]:

```
1 # Create the encoding transformer to preprocess the data
2
3 encoder = ColumnTransformer([
4     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len(
5         'onehot_encoder', OneHotEncoder(), categorical_cols) # encode cate
6     ], remainder='passthrough') # leave other columns unchanged
7
8 # Create the decision tree pipeline
9 dt_pipeline = Pipeline([
10     ('encoder', encoder), # encode and preprocess data
11     ('scaler', MinMaxScaler()), # apply feature scaling to normalized b
12     ('clf', DecisionTreeClassifier(random_state=42)) # decision tree cl
13 ])
14
15 # Fit the pipeline on the training data
16 dt_pipeline.fit(X_train, y_train)
17
18 # Generate predictions for the training and test data
19 y_train_pred_dt = dt_pipeline.predict(X_train)
20 y_test_pred_dt = dt_pipeline.predict(X_test)
21
22 # Print the classification reports for the training data
23 print('Classification report for decision tree w/out tuning training da
24 print(classification_report(y_train, y_train_pred_dt, target_names=['ne
25
26 # Print the confusion matrix for the training data
27 conf_mat_dt = confusion_matrix(y_train, y_train_pred_dt)
28 print('Confusion matrix for test data:')
29 print(conf_mat_dt)
30
31 # Check the AUC of predictions for training data
32 false_positive_rate_dt, true_positive_rate_dt, thresholds_dt = roc_curve(
33 roc_auc_dt = auc(false_positive_rate_dt, true_positive_rate_dt)
34 print('AUC:', roc_auc_dt)
35 print('\n')
36 print('=====')
37
38 # Print the classification reports for the test data
39 print('Classification report for decision tree w/out tuning test data:')
40 print(classification_report(y_test, y_test_pred_dt, target_names=['neut
41
42 # Print the confusion matrix for the test data
43 conf_mat_dt = confusion_matrix(y_test, y_test_pred_dt)
44 print('Confusion matrix for test data:')
45 print(conf_mat_dt)
46
47 # Check the AUC of predictions for test data
48 false_positive_rate_dt, true_positive_rate_dt, thresholds_dt = roc_curve(
49 roc_auc_dt = auc(false_positive_rate_dt, true_positive_rate_dt)
50 print('AUC:', roc_auc_dt)
```

Classification report for decision tree w/out tuning training data:

	precision	recall	f1-score	support
neutral or dissatisfied	1.00	1.00	1.00	40987
satisfied	1.00	1.00	1.00	30574
accuracy			1.00	71561
macro avg	1.00	1.00	1.00	71561
weighted avg	1.00	1.00	1.00	71561

Confusion matrix for test data:

```
[[40987  0]
 [ 1 30573]]
AUC: 0.9999836462353634
```

Classification report for decision tree w/out tuning test data:

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.95	0.95	13790
satisfied	0.93	0.94	0.93	10064
accuracy			0.94	23854
macro avg	0.94	0.94	0.94	23854
weighted avg	0.94	0.94	0.94	23854

Confusion matrix for test data:

```
[[13093 697]
 [ 642 9422]]
AUC: 0.9428321973596682
```

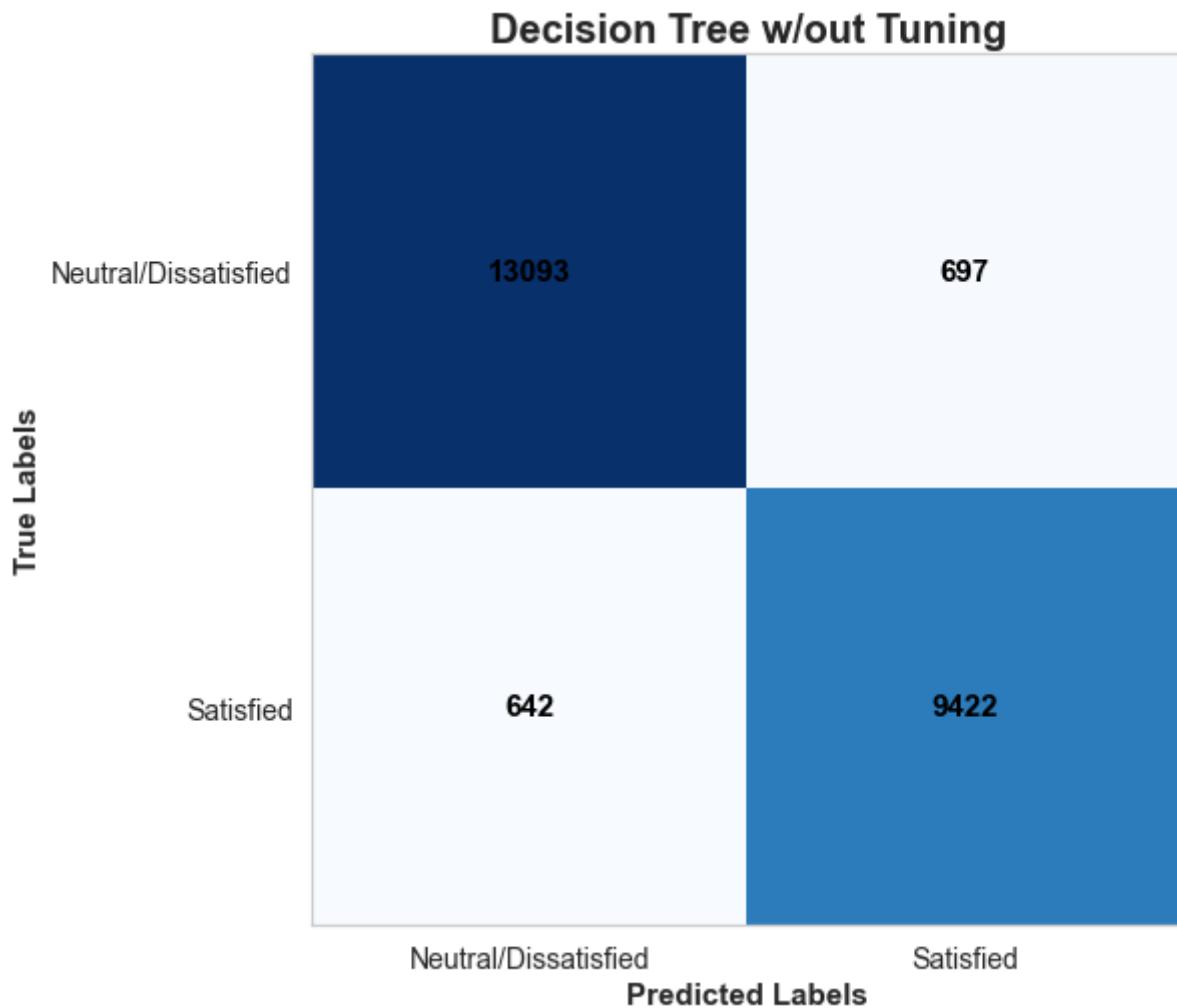
Observations:

- From the classification report for decision tree without tuning, it is observed that the model has an accuracy of 100% on the training data, which indicates that the model has overfit the training data. However, on the test data, the accuracy is 94%, which is a good result. The precision and recall for the 'neutral or dissatisfied' class are higher than those for the 'satisfied' class in both the training and test data.
- The confusion matrix for the training data shows that the model has classified all the observations correctly, which is not a good thing as it means that the model has overfit the training data. The confusion matrix for the test data shows that the model has classified 94% of the observations correctly, which is a good result.
- The AUC value for the test data is 0.94, which indicates that the model has good predictive power.
- In summary, the decision tree model without tuning has a good predictive power on the test data but has overfit the training data. Therefore, it is important to tune the model to improve its performance.

Confusion Matrix On Test Data

In [50]:

```
1 # get confusion matrix
2 cm_v = confusion_matrix(y_test, y_test_pred_dt)
3
4 # plot confusion matrix
5 fig, ax = plt.subplots(figsize=(8, 8))
6 ax.imshow(cm_v, cmap='Blues')
7 ax.grid(False)
8 ax.set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
9 ax.set_ylabel('True Labels', fontweight='bold', fontsize='15')
10 ax.set_title('Decision Tree w/out Tuning', fontweight='bold', fontsize='15')
11 ax.set_xticks([0, 1])
12 ax.set_yticks([0, 1])
13 ax.xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
14 ax.yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
15 ax.xaxis.set_tick_params(labelsize=14)
16 ax.yaxis.set_tick_params(labelsize=14)
17
18 for i in range(2):
19     for j in range(2):
20         ax.text(j, i, cm_v[i, j], ha='center', va='center', color='black')
```



Confusion matrix observations:

- The confusion matrix is a 2x2 matrix, where the rows represent the actual classes and the columns represent the predicted classes. In this case, the top left cell represents the number of true negatives, which is the number of times the algorithm correctly predicted a passenger would be neutral or dissatisfied (13,093 times). The top right cell represents the number of false positives, which is the number of times the algorithm predicted a passenger would be satisfied but they were actually neutral or dissatisfied (697 times).
- The bottom left cell represents the number of false negatives, which is the number of times the algorithm predicted a passenger would be neutral or dissatisfied but they were actually satisfied (642 times). Finally, the bottom right cell represents the number of true positives, which is the number of times the algorithm correctly predicted a passenger would be satisfied (9,422 times).
- Overall, the confusion matrix shows that the algorithm performed well, with a large number of

Decision Tree With No Tuning Results:

- **Test Data F1-Score For Satisfied Class:** 93%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **AUC Score:** 94%

Decision Tree with GridSearchCV

- GridSearchCV is a method used to tune hyperparameters in machine learning models. In the case of decision trees, hyperparameters such as the maximum depth of the tree and the minimum number of samples required to split an internal node can significantly affect the performance of the model.
- Using GridSearchCV, we can define a range of values for these hyperparameters and the method will perform an exhaustive search over all possible combinations of values to find the set of hyperparameters that gives the best performance on the training data.
- Once the best hyperparameters are identified, we can use them to train a decision tree model and evaluate its performance on the test data.

In [51]:

```

1 # Define the hyperparameters for tuning the model
2 params = {'clf__max_depth': [2, 3, 5, 7, 9, 13, 20],
3            'clf__min_samples_split': [2, 5, 10, 12, 15, 20, 25, 30, 35,
4               'clf__min_samples_leaf': [1, 2, 4]}
5
6 # Create the encoding transformer
7 encoder = ColumnTransformer([
8     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len
9     ('onehot_encoder', OneHotEncoder(), categorical_cols)
10 ], remainder='passthrough')
11
12 # Create the decision tree pipeline
13 dt_pipeline = Pipeline([
14     ('encoder', encoder),
15     ('scaler', MinMaxScaler()),
16     ('clf', DecisionTreeClassifier(random_state=42))
17 ])
18
19 # Create a grid search object
20 dt_gs = GridSearchCV(dt_pipeline, params, cv=5)
21
22 # Fit the grid search to the training data
23 dt_gs.fit(X_train, y_train)
24
25 # Print the best hyperparameters and best score
26 print("Best parameters: ", dt_gs.best_params_)
27 print("Best score: ", dt_gs.best_score_)
28
29 # Fit the pipeline on the training data with the best hyperparameters
30 dt_pipeline.set_params(**dt_gs.best_params_)
31 dt_pipeline.fit(X_train, y_train)
32
33 # Generate predictions for the training and test data
34 y_train_pred_dt_best = dt_pipeline.predict(X_train)
35 y_test_pred_dt_best = dt_pipeline.predict(X_test)

```

Best parameters: {'clf__max_depth': 13, 'clf__min_samples_leaf': 2, 'clf__min_samples_split': 30}
 Best score: 0.9498889106156041

- Now that we have the best parameters for tuning our decision tree, I will run another classification report on the training and test data to see if performed better on our data.

In [52]:

```
1 # Print the classification reports for the training and test data
2 print('Classification report for decision tree w/best parameters on tra
3 print(classification_report(y_train, y_train_pred_dt_best, target_names=
4
5 # Print the confusion matrix for the training data
6 conf_mat_dt_best_tr = confusion_matrix(y_train, y_train_pred_dt_best)
7 print('Confusion matrix for training data:')
8 print(conf_mat_dt_best_tr)
9
10 # Check the AUC of predictions
11 false_positive_rate_dt, true_positive_rate_dt, thresholds_dt = roc_curve(
12 roc_auc_dt = auc(false_positive_rate_dt, true_positive_rate_dt)
13 print('AUC:', roc_auc_dt)
14 print('\n')
15 print('=====')
16
17 print('Classification report for decision tree w/best parameters on tes
18 print(classification_report(y_test, y_test_pred_dt_best, target_names=[
19
20 # Print the confusion matrix for the test data
21 conf_mat_dt_best = confusion_matrix(y_test, y_test_pred_dt_best)
22 print('Confusion matrix for test data:')
23 print(conf_mat_dt_best)
24
25 # Check the AUC of predictions
26 false_positive_rate_dt, true_positive_rate_dt, thresholds_dt = roc_curve(
27 roc_auc_dt = auc(false_positive_rate_dt, true_positive_rate_dt)
28 print('AUC:', roc_auc_dt)
```

Classification report for decision tree w/best parameters on training data:

	precision	recall	f1-score	support
neutral or dissatisfied	0.94	0.98	0.96	40987
satisfied	0.97	0.92	0.95	30574
accuracy			0.96	71561
macro avg	0.96	0.95	0.96	71561
weighted avg	0.96	0.96	0.96	71561

Confusion matrix for training data:

```
[[40250 737]
 [2357 28217]]
AUC: 0.9524635211777697
```

Classification report for decision tree w/best parameters on test data:

	precision	recall	f1-score	support
neutral or dissatisfied	0.94	0.98	0.96	13790
satisfied	0.97	0.92	0.94	10064
accuracy			0.95	23854
macro avg	0.95	0.95	0.95	23854
weighted avg	0.95	0.95	0.95	23854

Confusion matrix for test data:

```
[[13474 316]
 [824 9240]]
AUC: 0.9476044252246104
```

Results on Training Data

- For the training data, the classification report shows a high level of precision and recall for both classes. The accuracy of the model was 0.96, indicating that it correctly predicted the class for 96% of the instances in the training data.
- The AUC score was 0.9524, which indicates that the model performed well in terms of correctly ranking the classes.
- The confusion matrix shows that the model predicted 40250 neutral or dissatisfied passengers and 28217 satisfied passengers correctly, but misclassified 737 neutral or dissatisfied passengers and 2357 satisfied passengers.

Results On Test Data:

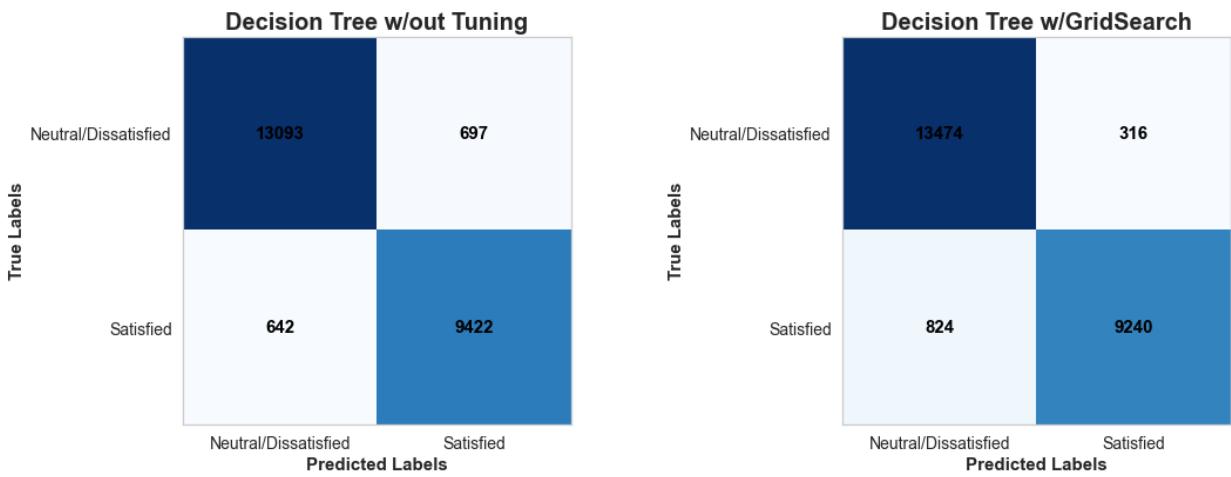
- For the test data, the classification report shows a high level of precision and recall for both classes as well, with an accuracy of 0.95.
- The AUC score was 0.9476, which indicates that the model performed well in terms of correctly ranking the classes.
- The confusion matrix shows that the model predicted 13474 neutral or dissatisfied passengers and 9240 satisfied passengers correctly, but misclassified 316 neutral or dissatisfied

passengers and 824 satisfied passengers.

Confusion Matrix of Test Data

In [53]:

```
1 # create a figure with two subplots
2 fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
3
4 # plot first confusion matrix
5 axs[0].imshow(cm_v, cmap='Blues')
6 axs[0].grid(False)
7 axs[0].set_xlabel('Predicted Labels', fontweight='bold', fontsize=15)
8 axs[0].set_ylabel('True Labels', fontweight='bold', fontsize=15)
9 axs[0].set_title('Decision Tree w/out Tuning', fontweight='bold', font
10 axs[0].set_xticks([0, 1])
11 axs[0].set_yticks([0, 1])
12 axs[0].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
13 axs[0].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
14 axs[0].xaxis.set_tick_params(labelsize=14)
15 axs[0].yaxis.set_tick_params(labelsize=14)
16
17 # add text boxes in the first confusion matrix
18 for i in range(2):
19     for j in range(2):
20         axs[0].text(j, i, cm_v[i, j], ha='center', va='center', color='
21
22 # plot second confusion matrix
23 conf_mat_dt_best = confusion_matrix(y_test, y_test_pred_dt_best)
24 axs[1].imshow(conf_mat_dt_best, cmap='Blues')
25 axs[1].grid(False)
26 axs[1].set_xlabel('Predicted Labels', fontweight='bold', fontsize=15)
27 axs[1].set_ylabel('True Labels', fontweight='bold', fontsize=15)
28 axs[1].set_title('Decision Tree w/GridSearch', fontweight='bold', font
29 axs[1].set_xticks([0, 1])
30 axs[1].set_yticks([0, 1])
31 axs[1].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
32 axs[1].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
33 axs[1].xaxis.set_tick_params(labelsize=14)
34 axs[1].yaxis.set_tick_params(labelsize=14)
35
36 # add text boxes in the second confusion matrix
37 for i in range(2):
38     for j in range(2):
39         axs[1].text(j, i, conf_mat_dt_best[i, j], ha='center', va='cent
40
41 # adjust spacing between subplots
42 plt.subplots_adjust(wspace=0.7)
43
44 #display the figure
45 plt.show()
46
```



Confusion Matrix Results:

- When comparing the two confusion matrixis, the Decision Tree using GridSearch has the better results for my goals of finding the passengers that were neutral/dissatisfied.
 - The model is better at predicting true negatives, and has a lower number of false positives. This means that the model is better at reducing the number of passengers that were falsely predicted at satisfied when they were actually neutral/dissatisfied.

Decision Tree (No Tuning) Results:

- Test Data F1-Score For Satisfied Class:** 93%
- Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- AUC Score:** 94%

Decision Tree With GridSearchCV Results:

- Test Data F1-Score For Satisfied Class:** 97%
- Test Data F1-Score For Neutral/Dissatisfied Class:** 94%
- AUC Score:** 95%

Comparing the two models:

- The decision tree with best parameters performs better than the decision tree without tuning. The accuracy score for the decision tree with best parameters on the test data is 0.95, while the accuracy score for the decision tree without tuning on the test data is 0.94. Additionally, the precision, recall, and f1-score for the decision tree with best parameters are higher than those for the decision tree without tuning.

Model 3: Random Forest

- Like the decision tree models, I will run a vanilla random forest classification model on the training and test data first.

- I will follow it up with a second random forest classification model using GridSearchCV to see if

Create Random Forest Pipeline

In [54]:

```
1 # Create the encoding transformer
2 # This transformer uses OrdinalEncoder for the ordinal columns and OneHotEncoder for the categorical columns
3 encoder = ColumnTransformer([
4     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len(categorical_cols), handle_unknown='ignore'),
5     ('onehot_encoder', OneHotEncoder(), categorical_cols)
6 ], remainder='passthrough')
7
8 #Define the pipeline
9 # This pipeline includes the encoder, scaler and the Random Forest Classifier
10 rf_pipeline = Pipeline([
11     ('encoder', encoder),
12     ('scaler', MinMaxScaler()),
13     ('clf', RandomForestClassifier(random_state=42))
14 ])
15
16 #Fit the pipeline on the training data
17 rf_pipeline.fit(X_train, y_train)
18
19 #Generate predictions for the training and test data
20 y_train_pred_rf = rf_pipeline.predict(X_train)
21 y_test_pred_rf = rf_pipeline.predict(X_test)
22
23 #Print classification report for training data
24 # This report shows precision, recall, f1-score and support for each class
25 print('\nClassification Report: **Random Forest With w/out Tuning On Training Data')
26 print(classification_report(y_train, y_train_pred_rf, target_names=['neutral', 'spam']))
27
28 #Display confusion matrix for Training data
29 # This matrix shows the actual and predicted classes
30 print('Confusion Matrix - Training Data:')
31 print(confusion_matrix(y_train, y_train_pred_rf))
32
33 # Check the AUC of predictions
34 # This metric measures the area under the curve of the Receiver Operating Characteristic
35 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_train_pred_rf)
36 roc_auc = auc(false_positive_rate, true_positive_rate)
37 print('AUC:', roc_auc)
38 print('\n')
39 print('=====')
40
41 #Print classification report for test data
42 # This report shows precision, recall, f1-score and support for each class
43 print('\nClassification Report: **Random Forest w/out Tuning On Test Data')
44 print(classification_report(y_test, y_test_pred_rf, target_names=['neutral', 'spam']))
45
46 #Display confusion matrix for Test data
47 # This matrix shows the actual and predicted classes
48 print('Confusion Matrix - Test Data:')
49 print(confusion_matrix(y_test, y_test_pred_rf))
50
51 # Check the AUC of predictions
52 # This metric measures the area under the curve of the Receiver Operating Characteristic
53 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_test_pred_rf)
54 roc_auc = auc(false_positive_rate, true_positive_rate)
55 print('AUC:', roc_auc)
```

Classification Report: **Random Forest With w/out Tuning On Training Data**

	precision	recall	f1-score	support
neutral or dissatisfied	1.00	1.00	1.00	40987
satisfied	1.00	1.00	1.00	30574
accuracy			1.00	71561
macro avg	1.00	1.00	1.00	71561
weighted avg	1.00	1.00	1.00	71561

Confusion Matrix - Training Data:

```
[[40986 1]
 [ 0 30574]]
AUC: 0.9999878010100763
```

=====

Classification Report: **Random Forest w/out Tuning On Test Data**

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.98	0.97	13790
satisfied	0.97	0.94	0.95	10064
accuracy			0.96	23854
macro avg	0.96	0.96	0.96	23854
weighted avg	0.96	0.96	0.96	23854

Confusion Matrix - Test Data:

```
[[13477 313]
 [ 646 9418]]
AUC: 0.9565566019246223
```

Classification Report On Training Data

- The classification report show perfect scores for precision, recall, f1-score, and accuracy. This is a sign of overfitting the training data.

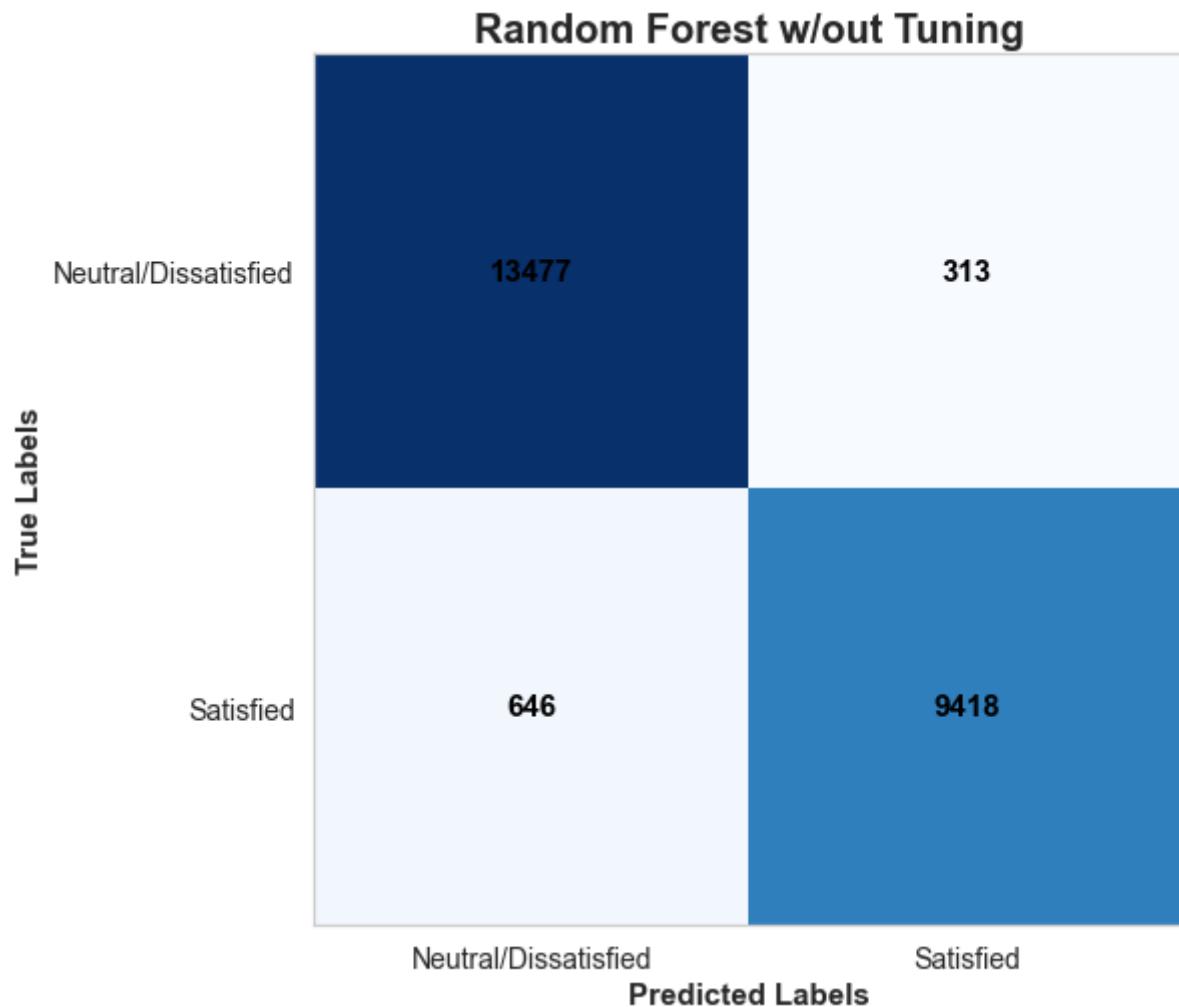
Classification Report On Test Data

- The model performed very well on the test data, with an accuracy of 0.96, precision of 0.95 and 0.97 for the two classes respectively, recall of 0.98 and 0.94 for the two classes, and F1-score of 0.97 and 0.95 for the two classes, respectively. The weighted average of precision, recall, and F1-score is 0.96, indicating very good performance. The confusion matrix shows that the model made 959 misclassifications out of 23854 samples, and the AUC is 0.956, indicating very good performance.

Confusion Matrix of Random Forest w/out Tuning

In [55]:

```
1 # get confusion matrix
2 cm_rfv = confusion_matrix(y_test, y_test_pred_rf)
3
4 # plot confusion matrix
5 fig, ax = plt.subplots(figsize=(8, 8))
6 ax.imshow(cm_rfv, cmap='Blues')
7 ax.grid(False)
8 ax.set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
9 ax.set_ylabel('True Labels', fontweight='bold', fontsize='15')
10 ax.set_title('Random Forest w/out Tuning', fontweight='bold', fontsize='15')
11 ax.set_xticks([0, 1])
12 ax.set_yticks([0, 1])
13 ax.xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
14 ax.yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
15 ax.xaxis.set_tick_params(labelsize=14)
16 ax.yaxis.set_tick_params(labelsize=14)
17
18 #Customize and create text boxes in the confusion matrix
19
20 for i in range(2):
21     for j in range(2):
22         ax.text(j, i, cm_rfv[i, j], ha='center', va='center', color='black')
```



Random Forest With No Tuning Results:

- **Test Data F1-Score For Satisfied Class:** 97%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **AUC Score :** 96%

Random Forest with GridSearchCV

- Use GridSearchCV to find the best hyperparameters to get the best model results.
- Due to a long run time compared to the GridSearch on the decision tree, I have removed min_samples_split because it is less critical than parameter.
- I also decreased the max_depth to help shorten the run time.

Create New Pipeline With GridSearch

In [56]:

```

1 # Define the hyperparameters for the tuning
2 params = {
3     'clf_criterion': ['gini', 'entropy'], # Split criterion for Decision Tree
4     'clf_n_estimators': [10, 12, 15, 20], # Number of trees in the forest
5     'clf_max_depth': [2, 3, 5, 7, 9], # Maximum depth of the tree
6     'clf_min_samples_leaf': [1, 2, 4] # Minimum number of samples required to split a node
7 }
8
9 # Create the encoding transformer
10 encoder = ColumnTransformer([
11     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len(categorical_cols)), categorical_cols), # One-hot encoding
12     ('onehot_encoder', OneHotEncoder(), categorical_cols) # One-hot encoding
13 ], remainder='passthrough')
14
15 # Define the pipeline
16 # This pipeline includes the encoder, scaler and the Random Forest Classifier
17 rf_pipeline = Pipeline([
18     ('encoder', encoder),
19     ('scaler', MinMaxScaler()),
20     ('clf', RandomForestClassifier(random_state=42))
21 ])
22
23 # Create a grid search object
24 rf_gs = GridSearchCV(rf_pipeline, params, cv=5) # 5-fold cross-validation
25
26 # Fit the pipeline on the training data
27 rf_gs.fit(X_train, y_train)
28
29 # Print the best hyperparameters and best score
30 print("Best parameters: ", rf_gs.best_params_) # Display the best hyperparameters
31 print("Best score: ", rf_gs.best_score_) # Display the best score
32
33 # Fit the pipeline on the training data with the best hyperparameters
34 rf_pipeline.set_params(**rf_gs.best_params_) # Set the best hyperparameters
35 rf_pipeline.fit(X_train, y_train)
36
37 # Generate predictions for the training and test data
38 y_train_pred_rf_best = rf_gs.predict(X_train) # Predictions on the training set
39 y_test_pred_rf_best = rf_gs.predict(X_test) # Predictions on the test set

```

Best parameters: {'clf_criterion': 'gini', 'clf_max_depth': 9, 'clf_min_samples_leaf': 1, 'clf_n_estimators': 20}
 Best score: 0.9412808824134166

In [57]:

```
1 # Print classification report for training data
2 # This report shows precision, recall, f1-score and support for each class
3 print('\nClassification Report: **Random Forest On Training Data with Best')
4 print(classification_report(y_train, y_train_pred_rf_best, target_names=
5
6 # Display confusion matrix for training data
7 print('Confusion Matrix - Training Data:'))
8 print(confusion_matrix(y_train, y_train_pred_rf_best))
9
10 # Check the AUC of predictions for the training data
11 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train)
12 roc_auc = auc(false_positive_rate, true_positive_rate)
13 print('AUC:', roc_auc)
14 print('\n')
15 print('=====')
16
17 # Print classification report for test data
18 # This report shows precision, recall, f1-score and support for each class
19 print('\nClassification Report: **Random Forest On Test Data with Best')
20 print(classification_report(y_test, y_test_pred_rf_best, target_names=
21
22 # Display confusion matrix for test data
23 print('Confusion Matrix - Test Data:'))
24 print(confusion_matrix(y_test, y_test_pred_rf_best))
25
26 # Check the AUC of predictions for the test data
27 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test)
28 roc_auc = auc(false_positive_rate, true_positive_rate)
29 print('AUC:', roc_auc)
30
```

Classification Report: **Random Forest On Training Data with Best Parameters**

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.95	0.95	40987
satisfied	0.94	0.93	0.93	30574
accuracy			0.94	71561
macro avg	0.94	0.94	0.94	71561
weighted avg	0.94	0.94	0.94	71561

Confusion Matrix - Training Data:

```
[[39080 1907]
 [ 2144 28430]]
AUC: 0.9416740548347174
```

Classification Report: **Random Forest On Test Data with Best Parameters**

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.95	0.95	13790
satisfied	0.93	0.93	0.93	10064
accuracy			0.94	23854
macro avg	0.94	0.94	0.94	23854
weighted avg	0.94	0.94	0.94	23854

Confusion Matrix - Test Data:

```
[[13113  677]
 [ 721 9343]]
AUC: 0.9396324797582636
```

Results on Training Data:

- The model achieved an overall accuracy of 0.94 on the training data. The precision and recall for the "neutral or dissatisfied" class are 0.95 and 0.95, respectively. This means that when the model predicted a customer is "neutral or dissatisfied", it was correct 95% of the time, and it identified 95% of the actual "neutral or dissatisfied" customers. The precision and recall for the "satisfied" class are 0.94 and 0.93, respectively. This means that when the model predicted a customer is "satisfied", it was correct 94% of the time, and it identified 93% of the actual "satisfied" customers.

Results On Test Data:

- The model achieved an overall accuracy of 0.94 on the test data. The precision and recall for the "neutral or dissatisfied" class are 0.95 and 0.95, respectively. This means that when the model predicted a customer is "neutral or dissatisfied", it was correct 95% of the time, and it identified 95% of the actual "neutral or dissatisfied" customers. The precision and recall for the

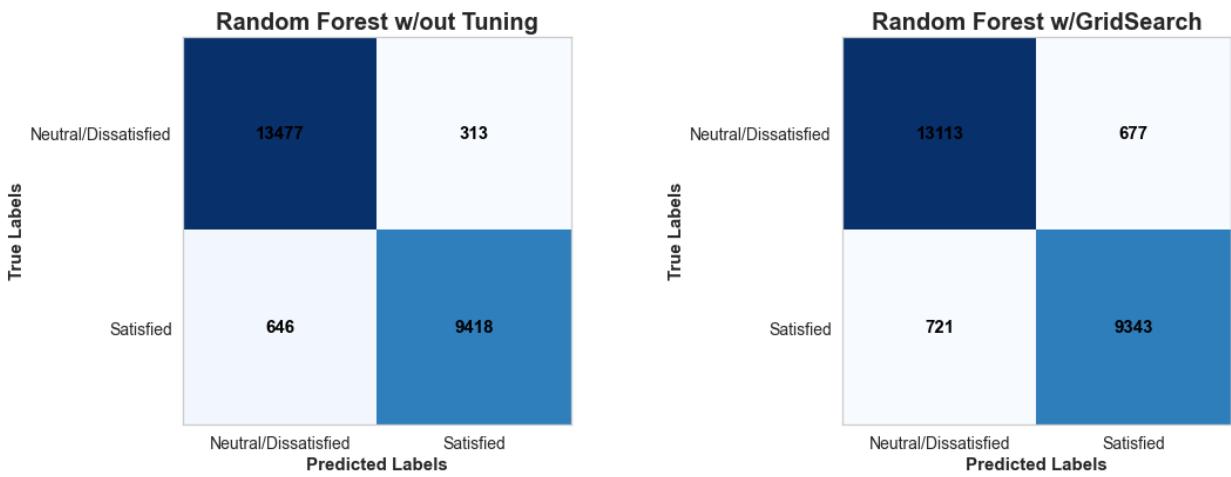
"satisfied" class are 0.93 and 0.93, respectively. This means that when the model predicted a customer is "satisfied", it was correct 93% of the time, and it identified 93% of the actual "satisfied" customers.

- Overall, the model's performance on the test data is very similar to that on the training data, indicating that the model did not overfit to the training data. The AUC score for both training and test data are close to 0.94, indicating that the model's ability to distinguish between positive and negative cases is good.

Confusion Matrix On Test Data

In [58]:

```
1 # Create two subplots
2 fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
3
4 # plot first confusion matrix
5 axs[0].imshow(cm_rfv, cmap='Blues') #make the color theme blue
6 axs[0].grid(False)
7 axs[0].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
8 axs[0].set_ylabel('True Labels', fontweight='bold', fontsize='15')
9 axs[0].set_title('Random Forest w/out Tuning', fontweight='bold', fonts
10 axs[0].set_xticks([0, 1]) # set the positions of the x-ticks
11 axs[0].set_yticks([0, 1]) # set the positions of the y-ticks
12 axs[0].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
13 axs[0].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
14 axs[0].xaxis.set_tick_params(labelsize=14) # set the font size of the x
15 axs[0].yaxis.set_tick_params(labelsize=14) # set the font size of the y
16
17
18 #customize the value inside the boxes
19 for i in range(2):
20     for j in range(2):
21         axs[0].text(j, i, cm_rfv[i, j], ha='center', va='center', color
22
23 # plot second confusion matrix
24 cm_rfgs = confusion_matrix(y_test, y_test_pred_rf_best)
25 axs[1].imshow(cm_rfgs, cmap='Blues') # make the color theme blue
26 axs[1].grid(False)
27 axs[1].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
28 axs[1].set_ylabel('True Labels', fontweight='bold', fontsize='15') # se
29 axs[1].set_title('Random Forest w/GridSearch', fontweight='bold', fonts
30 axs[1].set_xticks([0, 1]) # set the positions of the x-ticks
31 axs[1].set_yticks([0, 1]) # set the positions of the y-ticks
32 axs[1].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
33 axs[1].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
34 axs[1].xaxis.set_tick_params(labelsize=14) # set the font size of the x
35 axs[1].yaxis.set_tick_params(labelsize=14) # set the font size of the y
36
37 #Customize the values inside the boxes
38 for i in range(2):
39     for j in range(2):
40         axs[1].text(j, i, cm_rfgs[i, j], ha='center', va='center', colo
41
42 # adjust spacing between subplots
43 plt.subplots_adjust(wspace=0.7)
44
45 plt.show()
46
```



Confusion Matrix Results:

- Random forest models tend to perform better when data is not balanced. This is why the model with tuning did not do as well at predicting true negatives, false positives, and false negatives.
- Overall, the random forest model without tuning performed better at reducing the number of misclassifications with 959 out of 23854.

Random Forest (No Tuning) Results:

- **Test Data F1-Score For Satisfied Class:** 97%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **AUC Score:** 96%

Random Forest With GridSearchCV Results:

- **Test Data F1-Score For Satisfied Class:** 93%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **AUC Score:** 93%

Model 4: CatBoost Classifier

- For this model I will build two models using the catboost classifier.
 - The first model will be run with no tuning.
 - The second model I will use catboost with RandomSearchCV.
 - It is less computationally expensive than GridSearchCV.

CatBoost Pipeline

In [59]:

```
1 # Create the encoding transformer
2 encoder = ColumnTransformer([
3     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len,
4     ('onehot_encoder', OneHotEncoder(), categorical_cols) # One-hot enc
5 ], remainder='passthrough')
6
7 #Define the pipeline
8 cbc_pipeline = Pipeline([
9     ('encoder', encoder), # Encoding transformer
10    ('scaler', MinMaxScaler()), # Min-max scaling for all features
11    ('clf', ctb.CatBoostClassifier(verbose=False, random_state=42)) # C
12 ])
13
14 # Fit the pipeline to the training data
15 cbc_pipeline.fit(X_train, y_train)
16
17 # Make predictions on the training and test data
18 y_pred_cbc_train = cbc_pipeline.predict(X_train) # Predictions on the t
19 y_pred_cbc_test = cbc_pipeline.predict(X_test) # Predictions on the tes
20
21 # Display the classification report for the training and test data
22 print('Classification Report: **CatBoost w/out Tuning On Training Data*'
23 print(classification_report(y_train, y_pred_cbc_train, target_names=['n
24
25 #Display confusion matrix for training data
26 print('Confusion Matrix - Training Data:')
27 print(confusion_matrix(y_train, y_pred_cbc_train))
28
29 # Check the AUC of predictions
30 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train,
31 roc_auc = auc(false_positive_rate, true_positive_rate) # Compute AUC sc
32 print('AUC:', roc_auc)
33 print('\n')
34 print('=====')
35
36 # print classification report for the test data
37 print('\nClassification Report: **CatBoost w/out Tuning On Test Data**'
38 print(classification_report(y_test, y_pred_cbc_test, target_names=['neu
39
40 #Display confusion matrix for test data
41 print('\nConfusion Matrix - Test Data:')
42 print(confusion_matrix(y_test, y_pred_cbc_test))
43
44 # Check the AUC of predictions for test data
45 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
46 roc_auc = auc(false_positive_rate, true_positive_rate)
47 print('AUC:', roc_auc)
```

Classification Report: **CatBoost w/out Tuning On Training Data**

	precision	recall	f1-score	support
neutral or dissatisfied	0.97	0.99	0.98	40987
satisfied	0.98	0.95	0.97	30574
accuracy			0.97	71561
macro avg	0.98	0.97	0.97	71561
weighted avg	0.97	0.97	0.97	71561

Confusion Matrix - Training Data:

```
[[40533 454]
 [ 1450 29124]]
AUC: 0.9707486998515721
```

Classification Report: **CatBoost w/out Tuning On Test Data**

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.98	0.97	13790
satisfied	0.97	0.94	0.95	10064
accuracy			0.96	23854
macro avg	0.96	0.96	0.96	23854
weighted avg	0.96	0.96	0.96	23854

Confusion Matrix - Test Data:

```
[[13510 280]
 [ 644 9420]]
AUC: 0.9578524852113984
```

Results on Training Data:

- For the training data, the CatBoost model without tuning has a precision of 0.97 for neutral or dissatisfied and 0.98 for satisfied customers. It has a recall of 0.99 for neutral or dissatisfied and 0.95 for satisfied customers. The F1-score for neutral or dissatisfied is 0.98 and for satisfied customers is 0.97. The weighted average F1-score is 0.97 and the overall accuracy of the model is 0.97.
- AUC score = 97%

Results On Test Data:

- For the test data, the CatBoost model without tuning has a precision of 0.95 for neutral or dissatisfied and 0.97 for satisfied customers. It has a recall of 0.98 for neutral or dissatisfied and 0.94 for satisfied customers. The F1-score for neutral or dissatisfied is 0.97 and for satisfied customers is 0.95. The weighted average F1-score is 0.96 and the overall accuracy of the model is 0.96.
- AUC score = 96%

Confusion Matrix On Test Data w/out Tuning

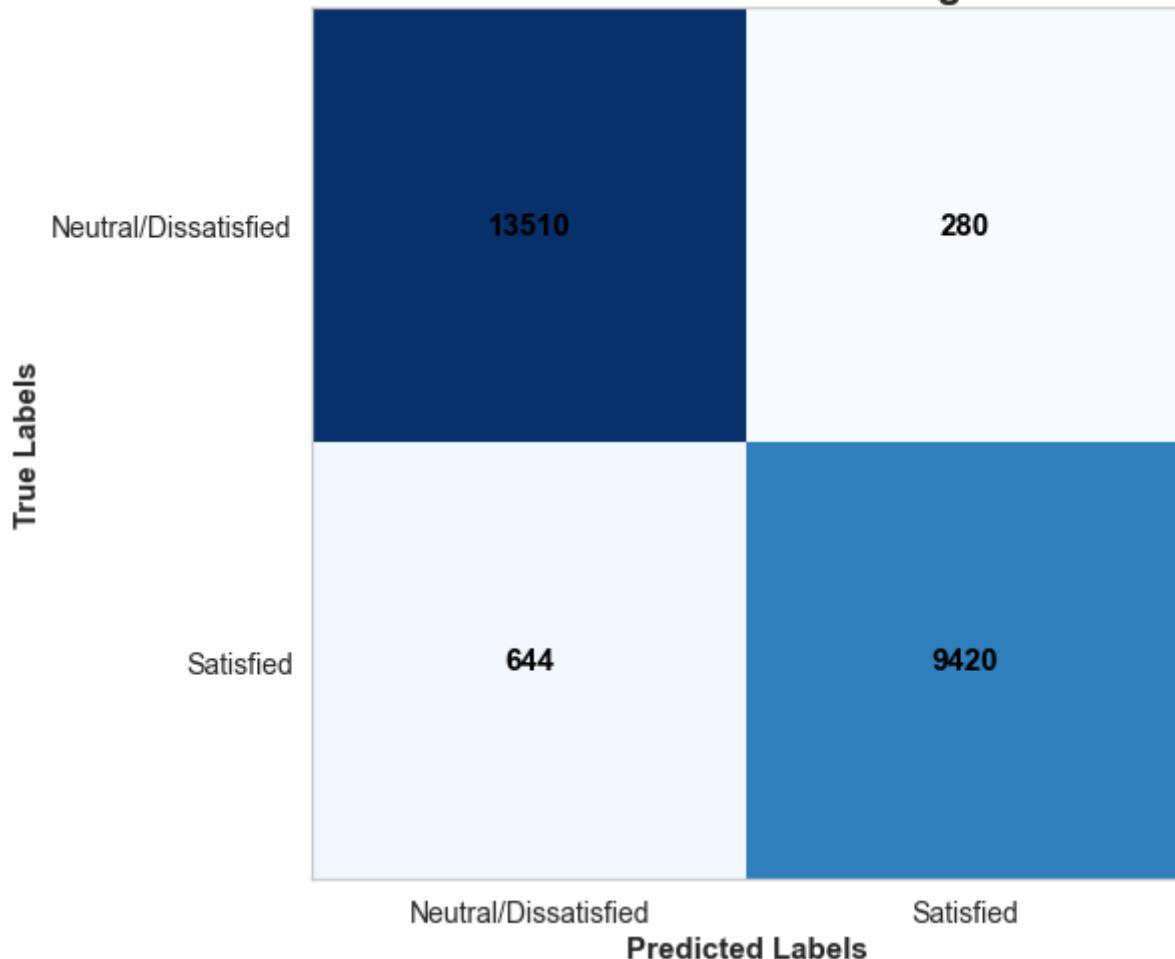
In [60]:

```

1 # get confusion matrix
2 cm_cbv = confusion_matrix(y_test, y_pred_cbc_test)
3
4 # plot confusion matrix
5 fig, ax = plt.subplots(figsize=(8, 8))
6 ax.imshow(cm_cbv, cmap='Blues') # Set a blue color scheme
7 ax.grid(False)
8 ax.set_xlabel('Predicted Labels', fontweight='bold', fontsize='15') #set x-axis label
9 ax.set_ylabel('True Labels', fontweight='bold', fontsize='15') # set yl
10 ax.set_title('Catboost w/out Tuning', fontweight='bold', fontsize='20')
11 ax.set_xticks([0, 1])
12 ax.set_yticks([0, 1])
13 ax.xaxis.set(ticks=[0, 1], ticklabels=('Neutral/Dissatisfied', 'Satisfi'))
14 ax.yaxis.set(ticks=[0, 1], ticklabels=('Neutral/Dissatisfied', 'Satisfi'))
15 ax.xaxis.set_tick_params(labelsize=14)
16 ax.yaxis.set_tick_params(labelsize=14)
17
18 # create a text boxes inside the confusion matrix
19 for i in range(2):
20     for j in range(2):
21         ax.text(j, i, cm_cbv[i, j], ha='center', va='center', color='bl

```

Catboost w/out Tuning



Confusion matrix observations:

- The model has correctly classified 13504 instances of neutral or dissatisfied customers and 9425 instances of satisfied customers. However, it has misclassified 286 instances of neutral or dissatisfied as satisfied and 639 instances of satisfied as neutral or dissatisfied.

CatBoost With No Tuning Results:

- **Test Data F1-Score For Satisfied Class:** 97%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **AUC Score:** 96%

CatBoost with RandomSearchCV

- RandomSearchCV is similar to GridSearchCV, however RandomSearchCV performs a random search over the parameter space, rather than an exhaustive search like GridSearchCV performs.
- RandomSearchCV is much less computationally expensive making it perform faster, especially for larger datasets.

Create New Pipeline With RandomSearch

In [61]:

```

1 # Define the hyperparameters to search
2 param_dist = {
3     'clf__iterations': sp_randint(500, 2000), # number of iterations for the classifier
4     'clf__learning_rate': [0.01, 0.1, 1], # learning rate for the model
5     'clf__max_depth': sp_randint(4, 10), # maximum depth of the trees used in the forest
6     'clf__random_strength': [0, 1, 3, 5, 7, 9], # random strength for the trees
7     'clf__l2_leaf_reg': [2, 4, 6, 8, 10] # L2 regularization for the model
8 }
9
10 # Create the encoding transformer
11 encoder = ColumnTransformer([
12     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len(categorical_cols)), categorical_cols),
13     ('onehot_encoder', OneHotEncoder(), categorical_cols) # one hot encoding for categorical features
14 ], remainder='passthrough')
15
16 # Define the pipeline
17 cbc_pipeline = Pipeline([
18     ('encoder', encoder), # encoding transformer
19     ('scaler', MinMaxScaler()), # min max scaling for the features
20     ('clf', ctb.CatBoostClassifier(verbose=False)) # catboost classifier
21 ])
22
23 # Create a randomized search object on the catboost classifier using the RandomizedSearchCV
24 cbc_rs = RandomizedSearchCV(cbc_pipeline, param_distributions=param_dist,
25                             cv=5, n_iter=20, n_jobs=-1, verbose=1, random_state=42)
26
27 # Fit the randomized search to the training data
28 cbc_rs.fit(X_train, y_train)
29
30 # Print the best hyperparameters and best score
31 print("Best parameters: ", cbc_rs.best_params_)
32 print("Best score: ", cbc_rs.best_score_)
33
34 # Fit the pipeline on the training data with the best hyperparameters
35 cbc_pipeline.set_params(**cbc_rs.best_params_)
36 cbc_pipeline.fit(X_train, y_train)
37
38 # Make predictions on the training and test data
39 y_pred_cbc_train_best = cbc_pipeline.predict(X_train)
40 y_pred_cbc_test_best = cbc_pipeline.predict(X_test)

```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.

[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 6.5min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 14.9min finished

Best parameters: {'clf__iterations': 1229, 'clf__l2_leaf_reg': 8, 'clf__learning_rate': 0.1, 'clf__max_depth': 5, 'clf__random_strength': 9}

Best score: 0.9594192115139458

In [62]:

```
1 # Display the classification report for the training and test data
2 print('Classification Report: **CatBoost On Training Data With Best Par
3 print(classification_report(y_train, y_pred_cbc_train_best, target_name
4
5 #Display confusion matrix for training data
6 print('Confusion Matrix - Training Data:')
7 print(confusion_matrix(y_train, y_pred_cbc_train_best))
8
9 # Check the AUC of predictions
10 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train
11 roc_auc = auc(false_positive_rate, true_positive_rate) #auc calculation
12 print('AUC:', roc_auc)
13 print('\n')
14 print('=====')
15
16 #print the classification report for the test data
17 print('\nClassification Report: **CatBoost On Test Data With Best Param
18 print(classification_report(y_test, y_pred_cbc_test_best, target_names=
19
20 #Display confusion matrix for test data
21 print('\nConfusion Matrix - Test Data:')
22 print(confusion_matrix(y_test, y_pred_cbc_test_best))
23
24 # Check the AUC of predictions for the test data
25 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
26 roc_auc = auc(false_positive_rate, true_positive_rate) # AUC score
27 print('AUC:', roc_auc)
```

Classification Report: **CatBoost On Training Data With Best Parameters**

	precision	recall	f1-score	support
neutral or dissatisfied	0.96	0.99	0.98	40987
satisfied	0.98	0.95	0.97	30574
accuracy			0.97	71561
macro avg	0.97	0.97	0.97	71561
weighted avg	0.97	0.97	0.97	71561

Confusion Matrix - Training Data:

```
[[40508 479]
 [ 1490 29084]]
AUC: 0.9697895745180163
```

Classification Report: **CatBoost On Test Data With Best Parameters**

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.98	0.97	13790
satisfied	0.97	0.94	0.95	10064
accuracy			0.96	23854
macro avg	0.96	0.96	0.96	23854
weighted avg	0.96	0.96	0.96	23854

Confusion Matrix - Test Data:

```
[[13495 295]
 [ 642 9422]]
AUC: 0.9574079769100671
```

Results on Training Data:

- Precision Score:** The precision for the "neutral or dissatisfied" class is 0.96, which means that 96% of the predictions for this class were correct.
- Recall Score:** The recall for this class is 0.99, which means that 99% of the actual "neutral or dissatisfied" samples were correctly identified.
- F1-Score:** The F1-score is the harmonic mean of precision and recall, and it is 0.98 for this class.
- Accuracy Score:** Accuracy is the proportion of correctly classified instances. On the training data, this model has an accuracy score of 0.97

Results On Test Data:

- Precision Score:** The precision of the model for neutral or dissatisfied customers is 95%, which means that out of all the customers predicted as neutral or dissatisfied, 95% were actually neutral or dissatisfied.
- Recall Score:** The recall for neutral or dissatisfied customers is also 98%, which means that out of all the actual neutral or dissatisfied customers, 98% were correctly predicted as neutral or dissatisfied.

- **F1-Score:** The F1-score for neutral or dissatisfied customers is 97%.
- **Accuracy:** Overall accuracy of 96%, which means that 96% of the predictions made by the model on the test set were correct.

Confusion Matrix On Test Data

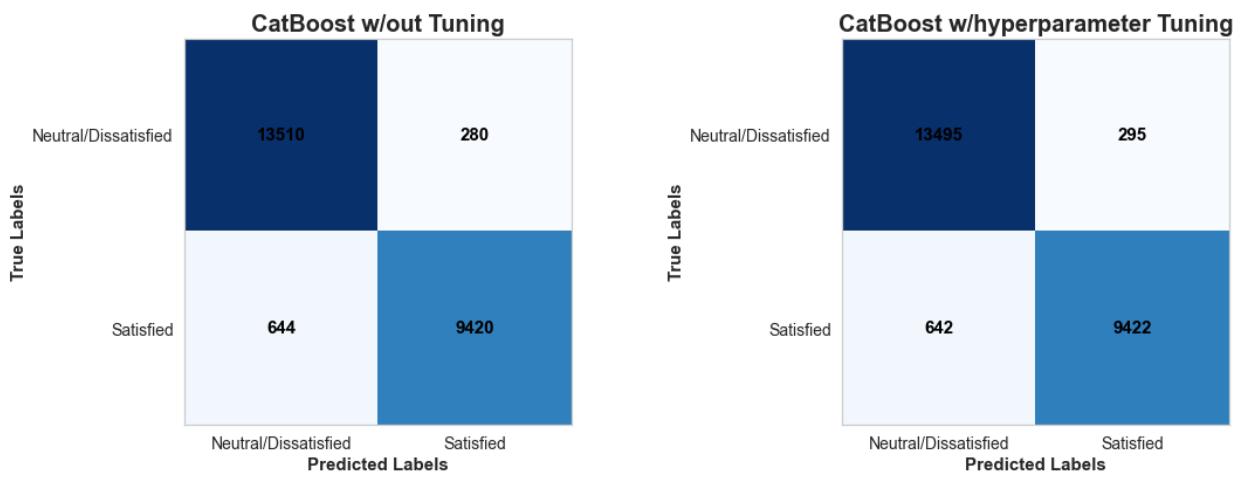
- Below is a comparison of the confusion matrixis of the test data of the model using catboost w/out tuning and the model using catboost w/gridsearch.

```
In [63]: 1 # adjust the spacing between the subplots
          2 plt.subplots_adjust(wspace=0.7)
          3
          4 plt.show()
```

<Figure size 432x288 with 0 Axes>

In [64]:

```
1 # create a figure with two columns and one row
2 fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
3
4 # plot first confusion matrix for CatBoost without tuning
5 axs[0].imshow(cm_cbv, cmap='Blues') # display the confusion matrix using
6 axs[0].grid(False) # turn off grid lines
7 axs[0].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
8 axs[0].set_ylabel('True Labels', fontweight='bold', fontsize='15') # set
9 axs[0].set_title('CatBoost w/out Tuning', fontweight='bold', fontsize='15')
10 axs[0].set_xticks([0, 1]) # set the positions of the x-ticks
11 axs[0].set_yticks([0, 1]) # set the positions of the y-ticks
12 axs[0].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
13 axs[0].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
14 axs[0].xaxis.set_tick_params(labelsize=14) # set the font size of the x-axis
15 axs[0].yaxis.set_tick_params(labelsize=14) # set the font size of the y-axis
16
17 # Add count values to the confusion matrix
18 for i in range(2):
19     for j in range(2):
20         axs[0].text(j, i, cm_cbv[i, j], ha='center', va='center', color='white')
21
22 # plot second confusion matrix for CatBoost with hyperparameter tuning
23 cm_cbrs = confusion_matrix(y_test, y_pred_cbc_test_best)
24 axs[1].imshow(cm_cbrs, cmap='Blues') # display the confusion matrix using
25 axs[1].grid(False) # turn off grid lines
26 axs[1].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
27 axs[1].set_ylabel('True Labels', fontweight='bold', fontsize='15') # set
28 axs[1].set_title('CatBoost w/hyperparameter Tuning', fontweight='bold', fontsize='15')
29 axs[1].set_xticks([0, 1]) # set the positions of the x-ticks
30 axs[1].set_yticks([0, 1]) # set the positions of the y-ticks
31 axs[1].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
32 axs[1].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfied'))
33 axs[1].xaxis.set_tick_params(labelsize=14) # set the font size of the x-axis
34 axs[1].yaxis.set_tick_params(labelsize=14) # set the font size of the y-axis
35
36 # add count values to the boxes of the confusion matrix
37 for i in range(2):
38     for j in range(2):
39         axs[1].text(j, i, cm_cbrs[i, j], ha='center', va='center', color='white')
40
41 # adjust the spacing between the subplots
42 plt.subplots_adjust(wspace=0.7)
43
44 plt.show()
```



Confusion Matrix Results:

- Compared to the model without tuning, the model with random search performed slightly better in terms of a higher true negative rate, and a lower false positive rate.
- Both models performed very well.

CatBoost (No Tuning) Results:

- Test Data F1-Score For Satisfied Class:** 97%
- Test Data F1-Score For Neutral/Dissatisfied Class:** 96%
- ROC AUC Score :** 96%

CatBoost With RandomSearchCV Results:

- Test Data F1-Score For Satisfied Class:** 97%
- Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- ROC AUC Score :** 96%

Model 5: XGBoost (No Tuning)

- Like the catboost, I will create two different pipelines:
 - The first pipeline will run XGBoost without any hyperparameter tuning.
 - The second pipeline will be integrated with RandomSearchCV.

XGBoost Pipeline

In [65]:

```
1 # Create the encoding transformer
2 # This transformer uses OrdinalEncoder for the ordinal columns and OneHotEncoder for the categorical columns
3 encoder = ColumnTransformer([
4     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len(categorical_cols), handle_unknown='ignore'),
5     ('onehot_encoder', OneHotEncoder(), categorical_cols) # One-hot encoding
6 ], remainder='passthrough')
7
8 # Define the pipeline
9 # This pipeline includes the encoder, scaler and the XGBoost Classifier
10 xgb_pipeline = Pipeline([
11     ('encoder', encoder), # Encoding transformer
12     ('scaler', MinMaxScaler()), # Min-max scaling for all features
13     ('clf', xgb.XGBClassifier(random_state=42))# xgboost Classifier with n_estimators=100
14 ])
15
16 # Fit the pipeline to the training data
17 xgb_pipeline.fit(X_train, y_train)
18
19 # Get feature importance
20 feat_importance = xgb_pipeline.named_steps['clf'].feature_importances_
21
22 # Get the 'encoder' step from the XGBoost pipeline
23 encoder = xgb_pipeline.named_steps['encoder']
24 # Get the one-hot encoded feature names for the categorical columns
25 onehot_cols = encoder.named_transformers_['onehot_encoder'].get_feature_names_out()
26 # Combine the one-hot encoded column names with the original ordinal column names
27 encoded_cols = list(onehot_cols) + ordinal_cols
28 # Combine the encoded column names with the numerical column names to get all columns
29 all_cols = encoded_cols + numerical_cols
30 # Create a dictionary mapping each feature name to its corresponding importance value
31 feature_importances = dict(zip(all_cols, feat_importance))
32 # Sort the feature importances dictionary by importance values in descending order
33 sorted_feature_importances = sorted(feature_importances.items(), key=lambda item: item[1], reverse=True)
34
35
36 # Make predictions on the training and test data
37 y_pred_xgb_train = xgb_pipeline.predict(X_train)
38 y_pred_xgb_test = xgb_pipeline.predict(X_test)
39
40 # Print classification report for training data
41 # This report shows precision, recall, f1-score and support for each class
42 print('Classification Report: **XGBoost w/out Tuning On Training Data**')
43 print(classification_report(y_train, y_pred_xgb_train, target_names=['Normal', 'Abnormal']))
44
45 # Display confusion matrix for training data
46 print('Confusion Matrix - Training Data:')
47 print(confusion_matrix(y_train, y_pred_xgb_train))
48
49 # Check the AUC of predictions for training data
50 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_pred_xgb_train)
51 roc_auc = auc(false_positive_rate, true_positive_rate)
52 print('AUC:', roc_auc)
53 print('\n')
54 print('=====')
55
56 # Print classification report for test data
57 # This report shows precision, recall, f1-score and support for each class
```

```

58 print('\nClassification Report: **XGBoost w/out Tuning On Test Data**')
59 print(classification_report(y_test, y_pred_xgb_test, target_names=['neu',
60
61 #Display confusion matrix for test data
62 print('\nConfusion Matrix - Test Data:')
63 print(confusion_matrix(y_test, y_pred_xgb_test))
64
65 # Check the AUC of predictions for the test data
66 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
67 roc_auc = auc(false_positive_rate, true_positive_rate)
68 print('AUC:', roc_auc)

```

Classification Report: **XGBoost w/out Tuning On Training Data**

	precision	recall	f1-score	support
neutral or dissatisfied	0.97	0.99	0.98	40987
satisfied	0.98	0.95	0.97	30574
accuracy			0.97	71561
macro avg	0.98	0.97	0.97	71561
weighted avg	0.97	0.97	0.97	71561

Confusion Matrix - Training Data:

```

[[40536  451]
 [ 1426 29148]]
AUC: 0.9711777871726218
=====
```

Classification Report: **XGBoost w/out Tuning On Test Data**

	precision	recall	f1-score	support
neutral or dissatisfied	0.96	0.98	0.97	13790
satisfied	0.97	0.94	0.95	10064
accuracy			0.96	23854
macro avg	0.96	0.96	0.96	23854
weighted avg	0.96	0.96	0.96	23854

Confusion Matrix - Test Data:

```

[[13502  288]
 [ 630 9434]]
AUC: 0.9582579684363799
```

Results on Training Data:

- On the training data, the classifier achieved an accuracy of 97%, with a precision of 0.97 for the 'neutral or dissatisfied' class and a precision of 0.98 for the 'satisfied' class. The recall for the 'neutral or dissatisfied' class was 0.99, while the recall for the 'satisfied' class was 0.95. The f1-score was 0.98 for the 'neutral or dissatisfied' class and 0.97 for the 'satisfied' class. The AUC was 0.97.

Results On Test Data:

- XGBoost is high performing model even without any tuning. The XGBoost classifier achieved an accuracy of 96%, with a precision of 0.96 for the 'neutral or dissatisfied' class and a precision of 0.97 for the 'satisfied' class. The recall for the 'neutral or dissatisfied' class was 0.98, while the recall for the 'satisfied' class was 0.94. The f1-score was 0.97 for the 'neutral or dissatisfied' class and 0.95 for the 'satisfied' class. The AUC was 0.96.
- Overall, the XGBoost classifier performed well on both the training and test data, achieving high accuracy and AUC scores with good precision and recall values.

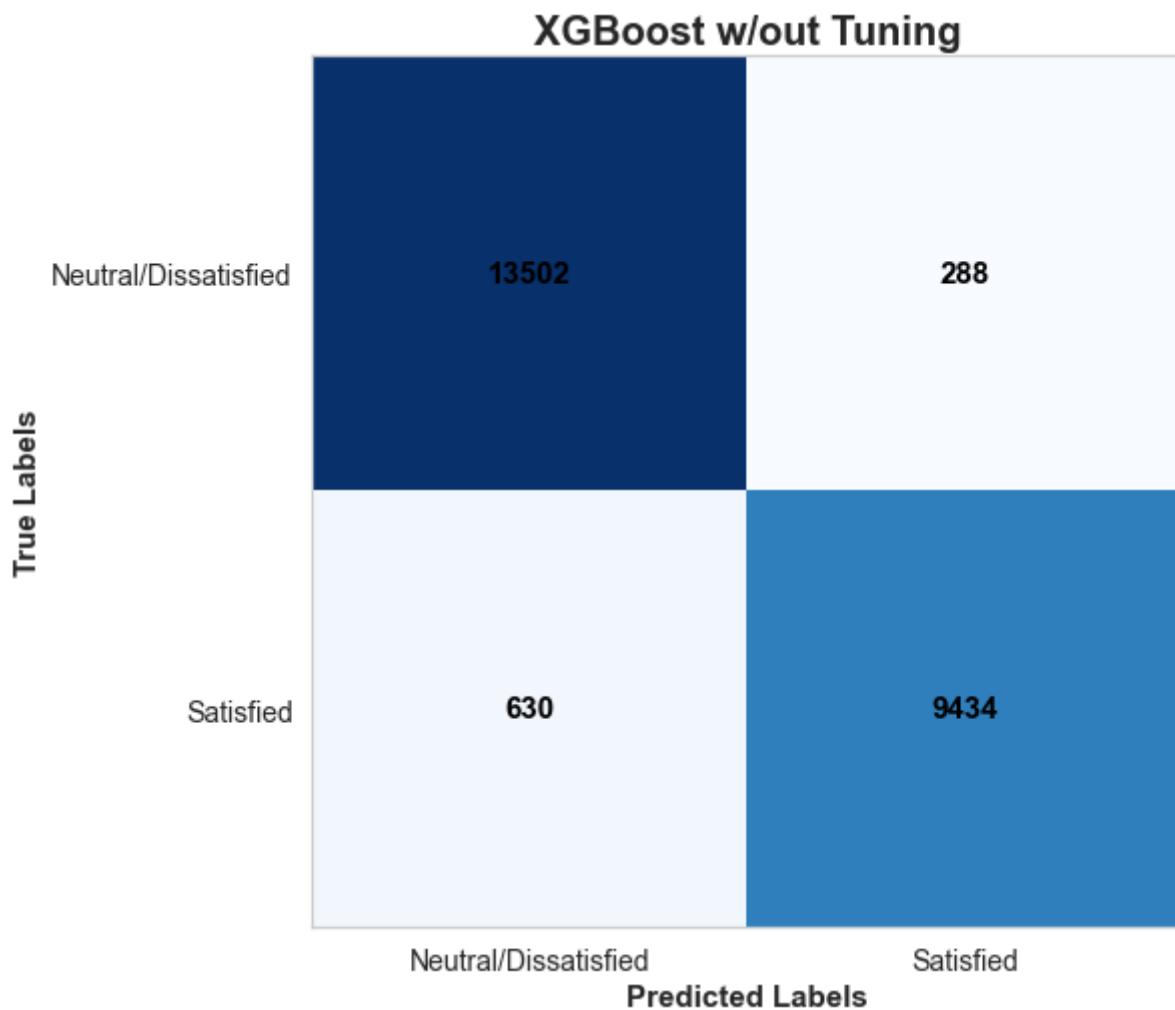
Confusion Matrix On Test Data

In [66]:

```

1 # get confusion matrix
2 cm_xgbv = confusion_matrix(y_test, y_pred_xgb_test)
3
4 # plot confusion matrix
5 fig, ax = plt.subplots(figsize=(8, 8))
6 ax.imshow(cm_xgbv, cmap='Blues') # Make the color theme blue
7 ax.grid(False)
8 ax.set_xlabel('Predicted Labels', fontweight='bold', fontsize='15') # s
9 ax.set_ylabel('True Labels', fontweight='bold', fontsize='15') # set y
10 ax.set_title('XGBoost w/out Tuning', fontweight='bold', fontsize='20')
11 ax.set_xticks([0, 1])
12 ax.set_yticks([0, 1])
13 ax.xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfi
14 ax.yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Satisfi
15 ax.xaxis.set_tick_params(labelsize=14)
16 ax.yaxis.set_tick_params(labelsize=14)
17
18 # Create a text box inside the confusion matrix
19 for i in range(2):
20     for j in range(2):
21         ax.text(j, i, cm_xgbv[i, j], ha='center', va='center', color='b

```



Confusion matrix observations:

- The XGBoost model with no tuning did well at reducing the number of missclassifications with 918 out of 23854.

XGBoost With No Tuning Results:

- **Test Data F1-Score For Satisfied Class:** 97%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 96%
- **AUC Score :** 96%

XGBoost with RandomSearchCV

Create New Pipeline With RandomSearch

In [67]:

```

1 # Define parameter grid
2 # dictionary of hyperparameters to be tuned using a randomized search.
3 param_grid = {
4     'clf__n_estimators': [50, 100, 200, 500, 1000], # Number of trees
5     'clf__learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3], # step size shrinkage
6     'clf__max_depth': [3, 4, 5, 6, 8, 10], # max depth of the trees
7     'clf__min_child_weight': [1, 3, 5, 7], # minimum sum of instance weight
8 }
9
10 # Create the encoding transformer
11 encoder = ColumnTransformer([
12     ('ordinal_encoder', OrdinalEncoder(categories=[[1, 2, 3, 4, 5]]*len(
13         'onehot_encoder', OneHotEncoder(), categorical_cols)
14     ], remainder='passthrough')
15 ]
16 # Define the pipeline
17 # This pipeline includes the encoder, scaler, and xgboost classifier
18 xgb_pipeline = Pipeline([
19     ('encoder', encoder),
20     ('scaler', MinMaxScaler()),
21     ('clf', xgb.XGBClassifier())
22 ])
23
24 # Create a randomized search object
25 xgb_rs = RandomizedSearchCV(xgb_pipeline,
26                             param_distributions=param_grid, #parameter
27                             cv=5, # 5 fold cross validation
28                             n_iter=20, # number of parameter settings to
29                             n_jobs=-1, # number of CPUs to use
30                             scoring='roc_auc', # Evaluation metric to use
31                             verbose=1, # Controls verbosity of the search
32                             random_state=42) # Random seed for reproduc
33
34 # Fit the randomized search to the training data
35 xgb_rs.fit(X_train, y_train)
36
37 # Print the best hyperparameters and score
38 print("Best hyperparameters:", xgb_rs.best_params_) # display best hyperparameters
39 print("Best score:", xgb_rs.best_score_) #display the best score
40
41 # Fit the pipeline on the training data with the best hyperparameters
42 xgb_pipeline.set_params(**xgb_rs.best_params_)
43 xgb_pipeline.fit(X_train, y_train)
44
45 # Make predictions on the training and test data
46 y_pred_xgb_train_best = xgb_pipeline.predict(X_train)
47 y_pred_xgb_test_best = xgb_pipeline.predict(X_test)
48

```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent worker s.
[Parallel(n_jobs=-1)]: Done 42 tasks | elapsed: 15.0min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 41.3min finished

```
Best hyperparameters: {'clf__n_estimators': 1000, 'clf__min_child_weight': 3, 'clf__max_depth': 10, 'clf__learning_rate': 0.01}
Best score: 0.993324951900043
```

In [68]:

```
1 # Display the classification report for training data
2 # This report shows precision, recall, f1-score and support for each class
3 print('Classification Report: **XGBoost On Training Data With Best Parameters')
4 print(classification_report(y_train, y_pred_xgb_train_best, target_names))
5
6 #Display confusion matrix for training data
7 print('Confusion Matrix - Training Data:')
8 print(confusion_matrix(y_train, y_pred_xgb_train_best))
9
10 # Check the AUC of predictions for training data
11 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train)
12 roc_auc = auc(false_positive_rate, true_positive_rate)
13 print('AUC:', roc_auc)
14 print('\n')
15 print('=====')
16
17 # Display the classification report for test data
18 # This report shows precision, recall, f1-score and support for each class
19 print('\nClassification Report: **XGBoost On Test Data With Best Parameters')
20 print(classification_report(y_test, y_pred_xgb_test_best, target_names))
21
22 #Display confusion matrix for test data
23 print('\nConfusion Matrix - Test Data:')
24 print(confusion_matrix(y_test, y_pred_xgb_test_best))
25
26 # Check the AUC of predictions for test data
27 false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test)
28 roc_auc = auc(false_positive_rate, true_positive_rate)
29 print('AUC:', roc_auc)
```

Classification Report: **XGBoost On Training Data With Best Parameters**

	precision	recall	f1-score	support
neutral or dissatisfied	0.96	0.99	0.98	40987
satisfied	0.99	0.95	0.97	30574
accuracy			0.97	71561
macro avg	0.98	0.97	0.97	71561
weighted avg	0.97	0.97	0.97	71561

Confusion Matrix - Training Data:

```
[[40581 406]
```

```
[ 1481 29093]]
```

AUC: 0.9708272846641711

=====

=

Classification Report: **XGBoost On Test Data With Best Parameters**

	precision	recall	f1-score	support
neutral or dissatisfied	0.95	0.98	0.97	13790
satisfied	0.97	0.94	0.95	10064
accuracy			0.96	23854
macro avg	0.96	0.96	0.96	23854
weighted avg	0.96	0.96	0.96	23854

Confusion Matrix - Test Data:

```
[[13521 269]
```

```
[ 646 9418]]
```

AUC: 0.9581519608803873

Results on Training Data:

- For the training data, the model achieved a precision of 0.96 for the 'neutral or dissatisfied' class and a precision of 0.99 for the 'satisfied' class. The recall values were 0.99 for the 'neutral or dissatisfied' class and 0.95 for the 'satisfied' class. The F1-score values were 0.98 for the 'neutral or dissatisfied' class and 0.97 for the 'satisfied' class.
- The auc score was 97%

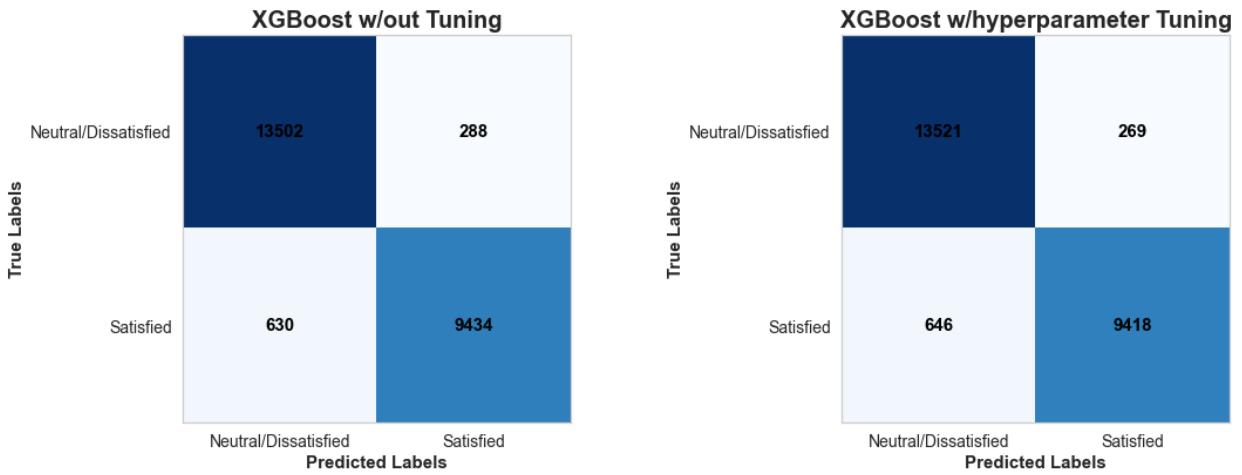
Results On Test Data:

- For the test data, the model achieved a precision of 0.95 for the 'neutral or dissatisfied' class and a precision of 0.97 for the 'satisfied' class. The recall values were 0.98 for the 'neutral or dissatisfied' class and 0.94 for the 'satisfied' class. The F1-score values were 0.97 for the 'neutral or dissatisfied' class and 0.95 for the 'satisfied' class.
- The auc score was 96%
- Overall the XGBoost model achieved an accuracy of 97% on the training data and an accuracy of 96% on the test data, which is a good performance.

Confusion Matrix On Test Data

In [69]:

```
1 # create a figure with two columns and one row
2 fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
3
4 # plot first confusion matrix
5 axs[0].imshow(cm_xgbv, cmap='Blues') # Blue color scheme
6 axs[0].grid(False)
7 axs[0].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
8 axs[0].set_ylabel('True Labels', fontweight='bold', fontsize='15') # se
9 axs[0].set_title('XGBoost w/out Tuning', fontweight='bold', fontsize='2
10 axs[0].set_xticks([0, 1])
11 axs[0].set_yticks([0, 1])
12 axs[0].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
13 axs[0].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
14 axs[0].xaxis.set_tick_params(labelsize=14)
15 axs[0].yaxis.set_tick_params(labelsize=14)
16
17 # create a text box inside the first confusion matrix
18 for i in range(2):
19     for j in range(2):
20         axs[0].text(j, i, cm_xgbv[i, j], ha='center', va='center', colo
21
22 # plot second confusion matrix
23 cm_xgbtrs = confusion_matrix(y_test, y_pred_xgb_test_best)
24 axs[1].imshow(cm_xgbtrs, cmap='Blues') # set blue color scheme
25 axs[1].grid(False)
26 axs[1].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
27 axs[1].set_ylabel('True Labels', fontweight='bold', fontsize='15') # se
28 axs[1].set_title('XGBoost w/hyperparameter Tuning', fontweight='bold',
29 axs[1].set_xticks([0, 1])
30 axs[1].set_yticks([0, 1])
31 axs[1].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
32 axs[1].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
33 axs[1].xaxis.set_tick_params(labelsize=14)
34 axs[1].yaxis.set_tick_params(labelsize=14)
35
36 # set text boxes inside the second confusion matrix
37 for i in range(2):
38     for j in range(2):
39         axs[1].text(j, i, cm_xgbtrs[i, j], ha='center', va='center', col
40
41 # adjust the spacing between the subplots
42 plt.subplots_adjust(wspace=0.7)
43
44 plt.show()
```



Confusion Matrix Results:

- Looking at the two confusion matrices, we can see that the XGBoost model with random search has 646 false negatives (i.e., 646 satisfied customers were incorrectly classified as dissatisfied) and 269 false positives (i.e., 269 dissatisfied customers were incorrectly classified as satisfied). On the other hand, the XGBoost model without tuning has 630 false negatives and 288 false positives. Therefore, the model with random search has a slightly higher false negative rate but a lower false positive rate compared to the model without tuning.
- Overall, the performance of both models is quite good and they are both capable of accurately predicting customer satisfaction based on the given features.

XGBoost (No Tuning) Results:

- **Test Data F1-Score For Satisfied Class:** 97%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **ROC AUC Score :** 96%

XGBoost With RandomSearchCV Results:

- **Test Data F1-Score For Satisfied Class:** 97%
- **Test Data F1-Score For Neutral/Dissatisfied Class:** 95%
- **ROC AUC Score :** 96%

Best Model:

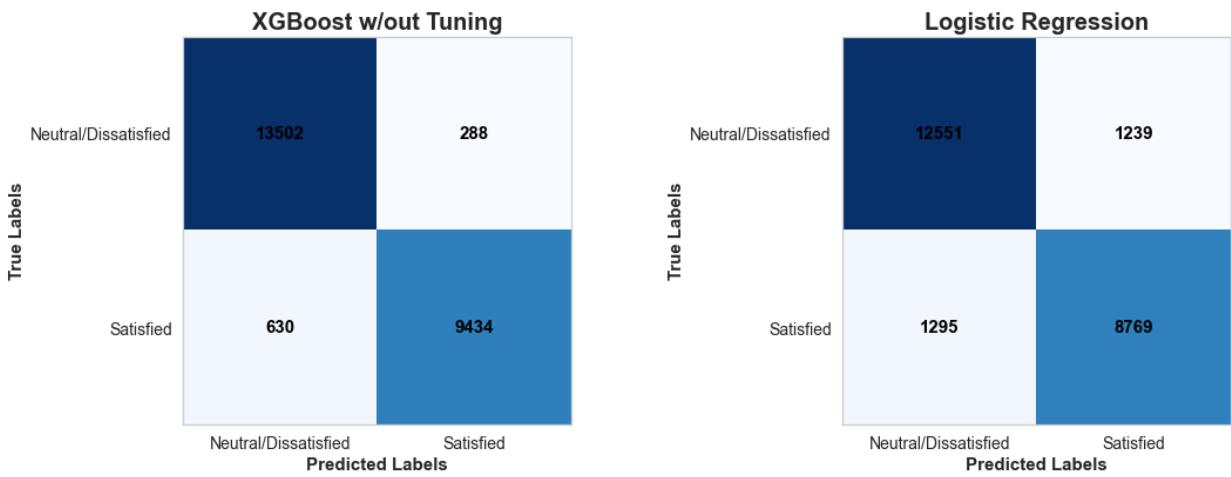
XGBoost without tuning

- Without tuning, XGBoost is already a high performing classifier. The F1 scores for both XGBoost models were the same.
- Both models performed the best in terms of reducing the number of misclassified passengers.
 - XGBoost w/out tuning = 918
 - XGBoost w/tuning = 915

- Both Models had high ROC AUC scores of 96% , making this model more capable of ranking ~~positive instances higher than negative instances~~

In [70]:

```
1 # create a figure with two columns and one row
2 fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
3
4 # plot first confusion matrix
5 axs[0].imshow(cm_xgbv, cmap='Blues') # Blue color scheme
6 axs[0].grid(False)
7 axs[0].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
8 axs[0].set_ylabel('True Labels', fontweight='bold', fontsize='15') # se
9 axs[0].set_title('XGBoost w/out Tuning', fontweight='bold', fontsize='2
10 axs[0].set_xticks([0, 1])
11 axs[0].set_yticks([0, 1])
12 axs[0].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
13 axs[0].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
14 axs[0].xaxis.set_tick_params(labelsize=14)
15 axs[0].yaxis.set_tick_params(labelsize=14)
16
17 # create a text box inside the first confusion matrix
18 for i in range(2):
19     for j in range(2):
20         axs[0].text(j, i, cm_xgbv[i, j], ha='center', va='center', colo
21
22 # plot second confusion matrix
23 cm_lg = confusion_matrix(y_test, y_test_pred)
24 axs[1].imshow(cm_xgbtrs, cmap='Blues') # set blue color scheme
25 axs[1].grid(False)
26 axs[1].set_xlabel('Predicted Labels', fontweight='bold', fontsize='15')
27 axs[1].set_ylabel('True Labels', fontweight='bold', fontsize='15') # se
28 axs[1].set_title('Logistic Regression', fontweight='bold', fontsize='20
29 axs[1].set_xticks([0, 1])
30 axs[1].set_yticks([0, 1])
31 axs[1].xaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
32 axs[1].yaxis.set(ticks=(0, 1), ticklabels=('Neutral/Dissatisfied', 'Sat
33 axs[1].xaxis.set_tick_params(labelsize=14)
34 axs[1].yaxis.set_tick_params(labelsize=14)
35
36 # set text boxes inside the second confusion matrix
37 for i in range(2):
38     for j in range(2):
39         axs[1].text(j, i, cm_lg[i, j], ha='center', va='center', color=
40
41
42 # adjust the spacing between the subplots
43 plt.subplots_adjust(wspace=0.7)
44
45 plt.savefig('images/confusion_compare.png', format='png')
46
47 plt.show()
```



Best Model and Baseline Model Comparison

- In comparing the two classification reports, it's clear that the XGBoost model outperforms the baseline Logistic Regression model in terms of accuracy, precision, recall, and F1-score for both training and test data.
- For the training data, the Logistic Regression model has an accuracy of 0.90, while the XGBoost model has an accuracy of 0.97. Similarly, for the test data, the Logistic Regression model has an accuracy of 0.89, while the XGBoost model has an accuracy of 0.96.
- The F1-scores for the Logistic Regression model are 0.91 and 0.88 for neutral or dissatisfied and satisfied passengers, respectively, on the test data. In contrast, the XGBoost model has F1-scores of 0.97 and 0.95 for the same classes. This shows that the XGBoost model has better overall performance in terms of balancing precision and recall.
- Additionally, the Area Under the Curve (AUC) scores for the Logistic Regression model are 0.8927 for the training data and 0.8907 for the test data. For the XGBoost model, the AUC scores are 0.9712 for the training data and 0.9583 for the test data. Higher AUC scores for the XGBoost model indicate that it has better classification performance overall.
- In terms of misclassification, the XGBoost model does a significantly better job at reducing the number of misclassified passengers.
 - XGBoost = **918**
 - Logistic Regression = **2534**
- In summary, the XGBoost model demonstrates superior performance over the baseline Logistic Regression model across all key evaluation metrics, making it the better choice for predicting airline passenger satisfaction.

XGBoost Feature Importance:

In [71]:

```
1 # Print out features with their corresponding importances
2 print("Feature importances:")
3 for feature, importance in sorted_feature_importances:
4     print(f'{feature}: {importance}')
```

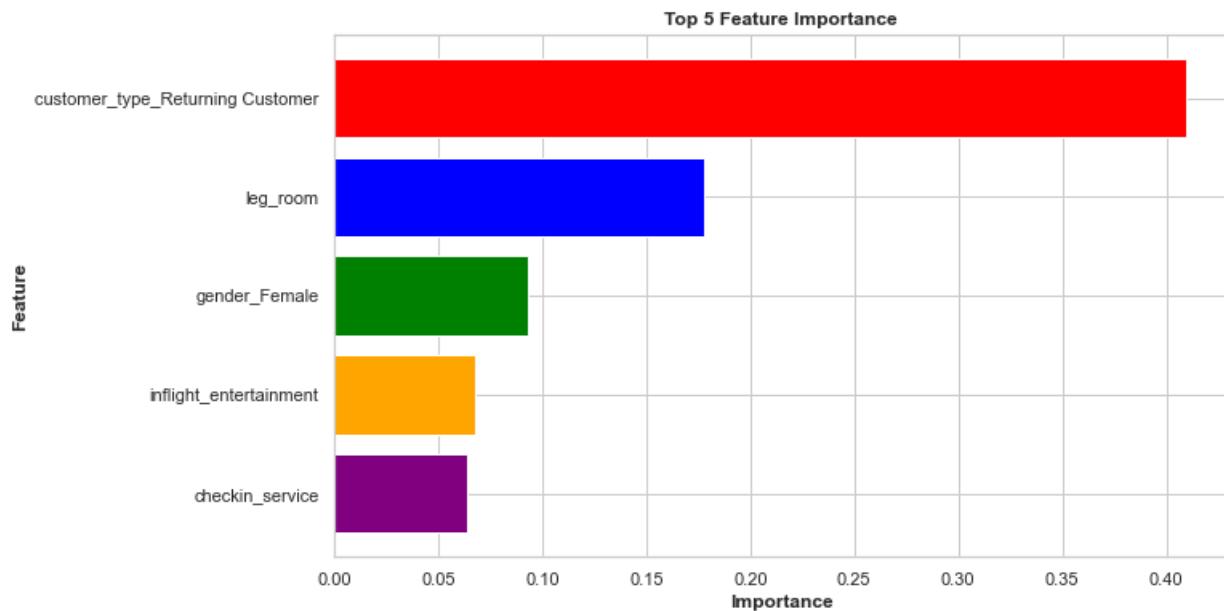
```
Feature importances:
customer_type_Returning Customer: 0.40922918915748596
leg_room: 0.17802201211452484
gender_Female: 0.09302108734846115
inflight_entertainment: 0.06800916790962219
checkin_service: 0.06374547630548477
type_of_travel_Personal Travel: 0.04073949158191681
inflight_wifi_service: 0.021656367927789688
class_Eco Plus: 0.017958687618374825
type_of_travel_Business travel: 0.017868647351861
food_and_drink: 0.016792047768831253
class_Eco: 0.0157735887914896
class_Business: 0.01570146158337593
ease_of_online_booking: 0.015274508856236935
gender_Male: 0.010434379801154137
inflight_service: 0.0037575974129140377
customer_type_First-time Customer: 0.0035373473074287176
flight_distance: 0.0033737767953425646
cleanliness: 0.0026141763664782047
online_boarding: 0.002490935381501913
seat_comfort: 0.0
on-board_service: 0.0
baggage_handling: 0.0
```

In [72]:

```

1 # Get the top 5 features from the sorted_feature_importances list
2 top_5_features = sorted_feature_importances[:5]
3
4 # Unzip the feature names and importance values into separate lists
5 feature_names, importance_values = zip(*top_5_features)
6
7 # Define a list of colors for each bar
8 colors = ['red', 'blue', 'green', 'orange', 'purple']
9
10 # Create a bar chart to visualize the top 5 features with different col
11 fig, ax = plt.subplots(figsize=(10,6))
12 plt.barh(feature_names, importance_values, color=colors)
13 ax.set_xlabel('Importance', fontweight='bold')
14 ax.set_ylabel('Feature', fontweight='bold')
15 ax.set_title('Top 5 Feature Importance', fontweight='bold')
16 plt.gca().invert_yaxis() # Reverse the order of the y-axis to show the
17
18 plt.savefig('images/feature_importance.png', format='png')
19 plt.show()
20

```



Top Five Important Features Results:

- **Customer_type_Returning Customer:** The most important feature is whether the passenger is a returning customer.
 - This suggests that customer loyalty is an important factor in determining customer satisfaction.
 - The company needs to reward their loyalty members with more perks.
 - Possibly increasing the amount of reward miles when a returning customer chooses to travel with Explorer Airlines.
- **Leg_room:** The second most important feature is whether the passenger had enough leg room during their flight.
 - It suggests that customers who had more leg room during their flight were more likely to have a positive experience with the airline. This makes sense as having enough leg room

- can greatly impact the overall comfort and satisfaction of a customer during a flight.
- A recommendation for this seems quite clear. Finding ways to increase leg room, even by an inch could increase passenger satisfaction.
 - Airlines that prioritize leg room can differentiate themselves from competitors and attract more customers who value comfort and quality of service.
 - Providing adequate leg room can also contribute to customer loyalty and repeat business, as customers are more likely to choose an airline that they know will provide a comfortable travel experience.
 - **Gender_Female:** The third most important feature in the model is gender, specifically whether the passenger is female.
 - This suggests that gender may play a significant role in predicting customer satisfaction in the airline industry.
 - Ensuring there is no favoritism toward one gender or the other is important for customer service.
 - A recommendation is to begin marketing campaigns geared toward female passengers, to see if airline passenger satisfaction increases.
 - **Inflight_entertainment** - This indicates that customers who had access to entertainment options during their flight were more likely to have a positive experience with the airline.
 - In-flight entertainment can provide a distraction during long flights and can greatly improve the overall experience of the customer.
 - This can also be a great opportunity where Explorer Airlines can promote their brand, through ads and short commercials.
 - My recommendation would be to educating passengers on how to use the mobile app for inflight entertainment leading up to the flight. Even if it a couple days before via text or email with a video attached to be helpful.
 - I recognize the fact that some customers choose not to download the app on their phones. To cater to these passengers, making sure there is a substantial amount of reading material could be helpful as well.
 - **Check-in_service** - This suggests that customers who had a good check-in experience were more likely to have a positive experience with the airline.
 - Check-in service can be a critical aspect of the travel experience, and a smooth and efficient process can greatly improve customer satisfaction.
 - My recommendation is to meet with the UX team to ensure the online check-in service is as user friendly as possible.
 - A second recommendation is to staff accordingly during the high traffic hours at the airport, to help with the check-in process run smoothly and as quickly as possible.

Conclusion

- My goal was to choose the best performing model not only by using evaluation metrics as key indicators but also by selecting the model that excelled at reducing the number of misclassified passengers.
- When Explorer Airlines rolls out its new marketing campaigns targeting dissatisfied customers, the company wants to ensure it reaches as many dissatisfied loyal customers as possible.
- Given that the most important feature in our analysis was customer_type_returning_customer, it is clear that customer loyalty is a critical factor for Explorer Airlines.

- By rewarding returning customers with additional perks and prioritizing their comfort and satisfaction, the company can increase customer loyalty and differentiate itself from competitors. This, in turn, could lead to increased revenue and profitability for the company in

Future Work

- As with any machine learning project, there is always room for improvement. Given more recent data and additional time for tuning and exploring other models, I am confident that the model's performance could be further improved.
- Once changes or improvements have been made to Explorer Airlines' business model, it would be valuable to re-analyze new survey data to determine if customer satisfaction levels have changed in response to these changes.
- While this analysis provides valuable insights into passenger satisfaction levels for Explorer Airlines, analyzing data from multiple airlines could provide a more comprehensive understanding of the factors that influence customer satisfaction. This would enable companies to identify industry-wide trends and best practices, and adapt their own strategies accordingly.